Workshops of the 39th IEEE Real-Time Systems Symposium

11th December, 2018

Edited under the guidance of Liliana Cucu-Grosjean, the RTSS2018 Workshops chair
Design of the logo by Amy Karns (Vanderbilt University)

Proceedings of the RTSS2018 Workshops
Table of contents

WMC 2018

Proceedings of the 6th International Workshop on Mixed Criticality Systems

Edited by

Jing LI and Zhishan GUO

Organisers

Program Chair

Jing Li and Zhishan Guo

Steering Committee

Robert Davis, Liliana Cucu-Grosjean, Claire Maiza, Sanjoy Baruah

Program Committee

Sebastian Altmeyer, University of Amsterdam, NL
Tam Chantem, Virginia Tech, USA
Arvind Easwaran, Nanyang Technological University, Singapore
Pontus Ekberg, Uppsala University, Sweden
Nathan Fisher, Wayne State University, USA
Nan Guan, Hong Kong Polytechnic University, China
Hyoseung Kim, University of California Riverside, USA
Jaewoo Lee, Chung-Ang University, Korea
Jinkyu Lee, Sungkyunkwan University (SKKU), Korea
Shen Li, IBM Thomas J. Watson Research Center, USA
Renato Mancuso, Boston University, USA
Dorin Maxim, LORIA lab, France
Geoffrey Nelissen, ISEP, Portugal
Dionisio de Niz, CMU, USA
Risat Mahmud Pathan, Chalmers University of Technology, Sweden
Linh Thi Xuan Phan, University of Pennsylvania, USA
Luca Santinelli, ONERA, France
Kecheng Yang, Texas State University, USA
Heechul Yun, University of Kansas, USA

Message from the Program Chairs

It is our pleasure to welcome you to the 6th International Workshop on Mixed Criticality Systems (WMC) at the Real-Time Systems Symposium (RTSS) in Nashville, Tennessee, USA on 11th December 2018.

The purpose of WMC is to share new ideas, experiences and information about research and development of mixed-criticality real-time systems. The workshop aims to bring together researchers working in fields relating to real-time systems with a focus on the challenges brought about by the integration of mixed-criticality applications onto single-core, multicore and many-core architectures. These challenges are cross-cutting. To advance rapidly, closer interaction is needed between the sub-communities involved in real-time scheduling, real-time operating systems / runtime environments, and timing analysis. This workshop aims to promote understanding of the fundamental problems that affect Mixed-Criticality Systems (MCS) at all levels in the software / hardware stack and crucially the interfaces between them.

WMC 2018 for the first time accepts the 2-page Journal-Never-Presented (JNP) papers. The JNP papers present relatively new journal papers in the scope of MCS that were accepted by major journals but not previously presented anywhere. This allows work that is relevant to the MCS community but might not be widely known to have the opportunity to be presented and discussed at the workshop.

For this sixth edition of the workshop, a total of 7 submissions were received. The review process involved 19 Program Committee members, with each submission receiving at least 5 reviews. We decided to accept 5 papers for presentation at the workshop, including 4 regular unpublished papers and 1 Journal-Never-Presented papers. We sincerely thank all the Program Committee members for their time and effort in the review process.

WMC 2018 would not be possible without the hard work of a number of people involved in the organization of RTSS 2018, including Robert I. Davis, Isabelle Puaut, Aniruddha Gokhale and Linda Buss. In particular, we would like to thank the RTSS 2018 workshops chair Liliana Cucu-Grosjean for her excellent organization and great support of the overall workshops program. We also thank the WMC Steering Committee for their guidance and suggestions throughout the preparation of WMC 2018.

Finally, we would like to thank all of the authors who submitted their work to WMC 2018, as well as all the keynote and invited talk speakers; without them, this workshop would not be possible. We wish you an interesting and exciting workshop and an enjoyable stay in Nashville, Tennessee, USA.

Zhishan Guo (University of Central Florida, USA)
Jing Li (New Jersey Institute of Technology, USA)
WMC 2018 Program Chairs

WMC 2018 Technical Program

WMC 2018 Keynote

Sanjoy Baruah

Mixed-Criticality Scheduling Theory : Scope, Promise and Limitations

BIO: Dr. **Sanjoy Baruah** is the professor at Washington University in St. Louis in September 2017. He was previously at the University of North Carolina at Chapel Hill (1999-2017) and the University of Vermont (1993-1999). He received his Ph.D. degrees from The University of Texas at Austin in 1993. His research interests and activities are in real-time and safety-critical system design, scheduling theory, resource allocation and sharing in distributed computing environments, and algorithm design and analysis. He is a Fellow of the IEEE, and the recipient of the 2014 Outstanding Technical Contributions and Leadership Award of the IEEE Technical Committee on Real-Time Systems.

ABSTRACT: Mixed-criticality Scheduling Theory arose in response to a very real and pressing need in safety-critical systems: how does one reconcile the goals of effective pre-run-time verification and resource-efficient implementation, despite the significant nondeterminism and uncertainty in the operating environments of many such systems? The theory has, over the past decade, evolved into an intellectually rich and exciting field of study; however, there is some concern that the developed theory does not match well with the requirements of actual system design. In this presentation I will attempt to explain (and justify) the dominant trends in mixed-criticality scheduling theory research. I will seek to characterize the problems in systems design and implementation that the developed theory is able to address, as well as the kinds of problems that it is cannot currently deal with, and will suggest some possible extensions to the theory that may broaden the scope of its applicability.

WMC 2018 invited talks

Mohamed Hassan (University of Guelph, Canada)
Title : **MPSoCs for Mixed-Criticality Systems: Challenges and Opportunities**

ABSTRACT. Real-time embedded systems are becoming ubiquitous in many emerging domains such as autonomous vehicles, healthcare, and drones. These domains pose two new major aspects that did not traditionally exist in real-time systems: 1) they are both computational and data intensive, and 2) they execute tasks with a mixed-criticality nature. Multiple-Processor Systems-on-Chip (MPSoCs) provide appealing platforms to address both aspects. Nonetheless, they come with their own challenges. In this talk, we discuss some of the opportunities that MPSoCs offer as well as their associated challenges.
BIO. Mohamed Hassan is an Assistant Professor at University of Guelph. Before Joining UoG, he worked as a SoC R&D engineer at Intel. He obtained his PhD from University of Waterloo in 2017 and MSc. From Cairo University in 2012. His research interests include real-time embedded systems, computer architecture, hardware validation and security.

Geoffrey Nelissen (CISTER ISEP, Portugal)
Title : **Mixed-Criticality Systems: What Is it Really?**

ABSTRACT. The talk will discuss some understandings on what can really be modeled with the Mixed-Criticality Systems model and how can we define a new terminology that is less confrontational/misleading when talking to industry.
BIO. Geoffrey Nelissen earned his MSc degree in Electrical Engineering at Université Libre de Bruxelles (ULB), Belgium in 2008. Then, he worked during four years as a PhD student in the PARTS research unit of ULB. In 2012, he received his PhD degree under the supervision of Professors Joël Goossens and Dragomir Milojevic, on the topic "Efficient Optimal Multiprocessor Scheduling Algorithms for Real-Time Systems". He is currently working at CISTER as a researcher scientist in the area of multiprocessor real-time scheduling theory. His research interests include real-time scheduling theory, real-time operating systems and multi-processors/multi-cores architectures.

Sathish Gopalakrishnan (University of British Columbia, Canada)
Title : **Mixed-Criticality Systems: Safety & Statistical Guarantees**

ABSTRACT. The talk will discuss whether we can provide statistical guarantees to ensure safety to Mixed-Criticality Systems and what are the existing techniques and challenges toward this goal.
BIO. Sathish Gopalakrishnan is an Associate Professor in the University of British Columbia's Electrical & Computer Engineering Department. He obtained a PhD in Computer Science and an MS in Applied Mathematics from the University of Illinois at Urbana-Champaign. Sathish's research interests span the design, modeling and analysis of computing systems. His work has been recognized with Best Paper Awards (IEEE RTSS 2004, IEEE Transactions on Industrial Informatics 2009). Sathish chaired the Vancouver Chapter of the IEEE Computer Society, is the Regional Coordinator for IEEE Regions 6 and 7 on the Computer Society's Geographic Unit Operations Committee.

Supporting Critical Modes in AirTight

J. Harbin, D. Griffin, A. Burns, I. Bate, R.I. Davis and L.S. Indrusiak
Department of Computer Science, University of York, UK.

***Abstract*—The AirTight protocol supports mixed criticality wireless traffic and temporal guarantees based on defined fault models. In some systems, following a catastrophic failure, it is necessary to communicate crucial data away from the site of the failure in order to better understand (post-hoc) the reasons why it occurred. To support this action it is necessary for a mode change request to be propagated to all the non-failed nodes in the system, and for these nodes to switch their behaviour so that the crucial data is given high priority in its use of the wireless network. This paper explains how AirTight can support such a critical mode change. A uni-cast protocol is utilised to flood the system with mode change messages, each node then locally prioritizes its use of the available bandwidth to support the defined UC (Ultra-Criticality) packet flows. An aircraft engine control scenario is used to motivate the requirements for the mode change protocol. Protocol-accurate simulations are then used to illustrate and evaluate the approach.**

I. Introduction

AirTight [2] is a wireless protocol (built upon the physical and MAC layers of IEEE 802.15.4) that supports mixed-criticality real-time traffic between computational nodes. With any wireless communication it is not realistic to assume fault-free behaviour. Rather, as in other considerations of fault tolerance, we require that certain levels of performance are delivered when the likelihood and severity of faults is bounded by what is referred to as a *fault model*. We assume that the physical layer of the protocol incorporates the usual methods of increasing resilience (for example spectrum spreading), AirTight therefore supports analysis that models the faults that manifest themselves as unacknowledged frame transmissions at the MAC layer.

Within AirTight the run-time behaviour is controlled via two-level scheduling. A system-wide slot table determines when each node can transmit and when each node must be open to receive (and on which channel). Local to each node is a fixed-priority scheduler that determines which packets to transmit when it has a transmission slot. An application packet (or flow) consists of a small sequence of frames; and it is assigned a criticality level [3]. As frames are transmitted each node keeps a count of the number of its transmission failures. When this number is below a defined limit, frames are simply resent. But if this limit is reached, a local critically mode change is made at the node and only the more critical packets are transmitted.

With a system defined to have two criticality levels, HI and LO, response-time analysis is used to verify that all packet deadlines are met if a lower threshold on the number of faults is satisfied. If this threshold is violated but a higher threshold is satisfied then the analysis will establish that all HI-critical packets are delivered by their deadlines. When there are currently no further frames to transmit then the node's failure count is re-set to zero.

In this paper we extend the scope for AirTight by defining the required behaviour of each node when there are more faults than the higher threshold specified, or when there is a functional mode change to the entire system brought about by a severe failure or attack. Although AirTight is a uni-cast protocol it uses a flooding scheme to communicate the requirement for this mode change to all non-failed nodes in the system. Each of these nodes then switches its local criticality mode to Ultra-Critical, UC. This will impact the set of local tasks that are executed and on the set of packets that are communicated. Within the context of the experienced failure, these tasks and packets may be 'new' (i.e. only occur in this UC mode), be existing HI-criticality tasks/packets or even be LO-criticality tasks/packets that have increased significance in the new mode.

II. Engine Malfunction Use Case

An aircraft engine is a harsh environment for electronics and wireless communication in that there are a lot of moving mechanical parts generating both interference and attenuating radio signals. Nevertheless, wireless sensors have two distinct advantages: (1) the sensors can be put deep inside the engine where it is not feasible to have cabling; and (2) it removes the weight and maintenance of cabling. The difficulty of maintenance may also mean that the designer may want to fit a number of replicas so replacement is not necessary. Current engines have a number of sensors. With a shift towards more intelligent control and monitoring, this number will grow. Internal to the engine there are failures that may affect the wireless communications but also may affect the requirements of the system. For example, in the case of a shaft break, there will be a significant amount of mechanical damage, which may cause nodes to fail and may lead to large pieces of material (including metal) being in unanticipated positions.

External to the engine there are a number of controlled interference sources, e.g. from the rest of the aircraft, and un-controlled interference sources, e.g. high-intensity radiated fields including lightning, mobile phones, laptops etc. This leads to complex fault behaviour that cannot be fully defined at design time. We therefore utilise a collection of fault models (one per criticality level) that are, in themselves, bounded. Finally, a number of parts of the overall aircraft system (and logistical support equipment on the ground) may want to use wireless communications and

as such the aircraft engine should be designed to share the same parts of the spectrum especially as the whole aircraft could have hundreds if not thousands of sensors.

A good example of the potential deployment of a wireless communication media is within an aircraft engine for the purposes of active health monitoring [4]. Figure 1 shows the communication graph (black lines) for a 25-node wireless network inspired by a possible engine monitoring system; it is clear that the topology of this example is a 5-node subsystem repeated 5 times. Actual data flows are shown as blue arrows. While this may not entirely represent how aircraft engines will ultimately use wireless communications, it is representative. An aircraft engine has a limited amount of space available to mount wireless sensors. In these places there are opportunities to use energy harvesting, e.g. using vibration, to power the nodes. Therefore in these locations there will be a number of smart sensors (i.e. nodes) monitoring different properties of the engine which will then communicate with the rest of the engine via a signal concentrator. Then, in one central location there will be the traditional aircraft engine controls system (termed a FADEC – Full Authority Digital Control system) that takes all the signals, provides the primary control and monitoring, and importantly provides the links to the Avionics Full-Duplex Switched Ethernet (AFDX), i.e. the communications to the rest of the aircraft.

We have used this 5-node subsystem to illustrate the analysis associated with AirTight [2], and have validated this analysis using a prototype network of 5 IEEE 802.15.4 compliant nodes. We also used a protocol-accurate in-house simulator to evaluate AirTight's performance and scalability over the complete 25-node network. In total this network has 55 packet flows mapped to the 25 nodes; 25 of these flows are defined to be of HI-criticality and 30 of LO-criticality.

In this paper we will use this example to illustrated how AirTight supports the need for the criticality mode change that would follow a significant mechanical failure, e.g. a shaft break. A shaft break (or similar catastrophic failure) is a very rare but not unknown event[1]. It is an interesting example within the context of this work for two reasons. Firstly, as the engine is effectively damaged beyond repair then this event is rarely, and certainly not comprehensively, investigated on a test rig which means if/when it does happen for real there is a strong desire to get as much engine data as possible into long-term storage for later diagnosis and understanding. Secondly, from the point at which the shaft break is detected, more complex control algorithms are performed for a limited amount of time but this extra functionality can be at the expense of some of the "normal" functionality including that which is normally HI-criticality. For these reasons, the shaft break mode change can be modelled as follows: (i) the amount of data being communicated from the smart sensors to long-term storage is increased by a factor of, perhaps, 5; (ii) the time for which the best-effort communications must be maintained is, for example, 20 seconds; (iii) a percentage of nodes will be randomly lost, e.g. 10%; (iv) as some nodes may be lost, including those responsible for signal concentration and communications to the airframe, some signals may need to be sent to a number of sinks instead of just one; and (v) a percentage of the "normal" HI-criticality messages will become LO-criticality, e.g. 50%.

[1]An example is reported in https://www.nbcnews.com/storyline/airplane-mode/faa-orders-a380-engine-inspections-after-midair-failure-emergency-landing-n810341.

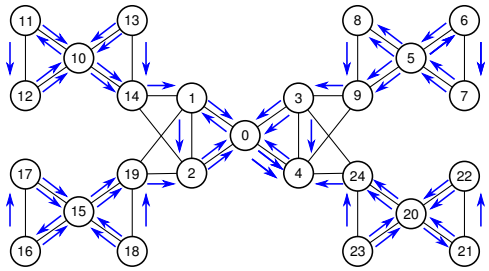


Fig. 1. Communication Graph of a 25 node Health Monitoring System

III. Overview of AirTight

We assume a distributed system of nodes that can each perform any combination of executing tasks, producing/consuming data from sensors/tasks, writing to actuators and relaying data packets to and from other nodes. The AirTight protocol has the following basic properties (most of them inherited from the parent standard IEEE 802.15.4):

- Peer-to-peer packet-switching communication between tasks/nodes is the normal use of the network. Packets are sent as one or more *frames*. Each successful frame transmission is always acknowledged by the receiver through the transmission of a short ACK frame.
- Multi-hop routing is required due to the limited transmission range of each node.
- Buffers exist on each node to store frames in transit (the size of the buffers required on each node can be determined during the offline schedulability analysis).
- Nodes have line power or local harvesting, so energy efficiency/battery life is not a limiting concern.
- Multiple frequency bands (channels) are available in IEEE 802.15.4 (up to 16 in the 2.4GHz band) but a node can only use one channel at a time.
- Node communications are represented by two graphs: the *communications graph* and the *interference graph*:
  - The communications graph $\mathbf{C}$: if there is an edge from $A \rightarrow B$ in $\mathbf{C}$, then the two nodes can communicate directly. This is required to be a symmetric graph due to the necessity for an acknowledgement to be returned to the sender, so $A \rightarrow B$ implies $B \rightarrow A$.
  - The interference graph $\mathbf{I}$: if there is an edge from $A \rightarrow B$ in $\mathbf{I}$, then a transmission from A will prevent B from receiving a frame from any node other than A on that channel at that time.

Note $\mathbf{C}$ is a subgraph of $\mathbf{I}$: if $A \rightarrow B$ is in $\mathbf{C}$ then it will also be in $\mathbf{I}$.

It is assumed that the packets to be communicated have tight timing constraints (i.e. deadlines). We also require that the system supports applications of different levels of criticality.

AirTight is designed to balance efficiency and flexibility. At the system level, its media access control is table-driven, but at the node level it uses criticality-aware priority-based frame scheduling. The protocol is based around the repeated application of the slot tables which, in time, define the activities of each node – either transmission or reception on that channel, or null meaning no usage. The slot (or scheduling) table (ST) consists of a series of slots. Each slot is assigned to a node and can be used by that node to send a single data frame on a designated channel. The slot also accommodates the ACK frame of the respective receiver.

At each node, local scheduling decisions are made to manage the use of the node's slot allocation. We employ a fixed-priority scheme. A set of FIFO queues (buffers), one per priority level, are used to hold the frames that need to be transmitted. Each normal flow has a unique priority and hence a specific buffer. The frames from the same flow are stored in the buffer in FIFO order. Whenever the node has a slot available, it transmits the first frame in the highest priority non-empty buffer. If an ACK is received the frame is removed from the buffer; if no ACK is received, then the frame remains in the buffer and is a candidate for re-transmission when the next available slot for that node becomes available.

AirTight is thus a two level protocol. A collection of slot tables defines the usage of the wireless media. Each slot in a table defines whether the node can transmit in that slot (and on which channel if more than one channel is used), or whether it should listen in that slot (and on which channel), or whether it is off-duty. The collection of tables reflects the properties of the communication and interference graphs.

The fundamental time unit of AirTight is the duration (S) of a slot – the time it takes to communicate a single frame of data and receive an ACK for that frame. In our prototype implementation [2] a slot length of 10ms has been achieved. All parameters of the application, the communication media and the environment (e.g. the usual T_i, C_i, D_i, table length, fault models, etc.) are expressed as an integer number of slot times.

A schedulable AirTight network supporting mode changes is intended to support the following requirements:

- If there are no faults experienced by the system then all packets will meet their deadlines.
- If the faults experienced by the system are no worse than that implied by the LO-criticality fault model then all packets will meet their deadlines. This is defined to be the LO-criticality mode.
- If the faults experienced by the system are no worse than that implied by the HI-criticality fault model then all HI-criticality packets will meet their deadlines. This is defined to be the HI-criticality mode.
- If the faults experienced by the system are worse than that implied by the HI-criticality fault model then we assume that this level of faults implies a permanent degradation to the network and/or the control system it is supporting. This is defined to be the Ultra-Critical (UC) mode, and is the focus of this paper.

For this mixed-criticality behaviour response-time analysis has been developed [2] that can be used to verify an application. This analysis is itself based upon the approach developed for mixed criticality task scheduling [1]; it is not repeated here due to space limitations.

The application's characteristics, together with the per channel interference and communication graphs, and the analysis developed for AirTight, are the inputs required to construct the per channel slot tables. The simplest slot table is one that has a single slot per node (with some slots being used by more than one node if they are not linked in the interference graph). More complex slot tables can be constructed, via search techniques such as the use of Genetic Algorithms that also take task placement and routing into account. The use of these techniques to construct optimal, or near optimal, slot tables forms part of future work and is not considered further here.

IV. Supporting Critical Mode Changes

For ease of presentation we will assume that our system is multi-hop and multi-domain, but single channel.

In the LO- or HI-criticality (i.e. not UC) mode of operations each node (n_i) will have a set of other nodes that it sends messages to. Let this set be represented by P_i (for partners). Clearly each member of P_i is linked to n_i in the communication graph, $\mathbf{C}$. Let the larger set of nodes that n_i could communicate with be denoted by P_i^+. So P_i^+ contains all the partners of n_i in $\mathbf{C}$.

A mode change is triggered within n_i by either an application task that has identified a severe physical failure, or attack, or the AirTight protocol stack having monitored more frame communication failures than can be tolerated in the HI-criticality mode; let this value be represented by G_{HI}. Node n_i also undergoes a mode change if it receives an authenticated 'mode-change' packet from another node. This packet is then passed on to all members of P_i^+.

In the protocol described in this paper, the slot table does not change when the node switches to the UC mode. It is possible to envisage a protocol in which a different slot table becomes more appropriate in the UC mode. But to coordinate the simultaneous switching of all nodes to a new table is not without considerable difficulty. We therefore explore in this work the expressive power of an approach that retains the same slot table in this UC mode. This has the advantage that the mode change can be communicated across the system without the need for coordination. Of course the initial (offline) construction of the slot table could take into account the needs of the UC mode. For example, a node that does not transmit any packets under normal operation, and hence does not require a slot in the table,

could be assigned a slot so that it could contribute to the communication of the critical mode change request.

Having recognised the need for a system-wide mode change the node follows the following (initial phase) protocol:

- A single 'mode-change' packet of the highest local priority is queued (buffered) ready to be sent to all members of P_i^+. A *distribution queue* is initialised, containing the members of P_i^+. This distribution queue may optionally be sorted in such a way as to direct the mode change message more quickly in a specific direction.
- When transmitting the packet, its next-hop destination is set to the node at the head of the distribution queue. If the transmission is acknowledged successfully, the entry at the head of the distribution queue is removed. If the distribution queue is empty, then all peers have been informed of the mode change, and the mode change packet is deleted from its buffer.
- If a frame from one of these packets fails to be acknowledged then the associated packet is not removed from the buffer – this is the usual behaviour for AirTight.
- If any frame fails to be sent to the same next-hop destination G_{HI} times (determined by the lack of an acknowledgement), the next-hop destination is removed from the distribution queue – the wireless link or designated node is assumed to be permanently broken as a result of the primary cause of the mode change.
- If node n_i receives a 'mode-change' packet from n_j while it is already distributing its mode change, then n_j is removed from the distribution queue (if it is currently present) – clearly n_j does not need to be informed of the mode change.

The above flooding behaviour ensures that all non-failed nodes receive the mode change request within a bounded period of time. The analysis developed for AirTight [2] can be used to compute this value for various system failure scenarios (an example is provided in the Evaluation section).

In the second phase of the protocol the packets associated with the UC mode are queued and transmitted. These packets arise from:

- Packets that are only sent in the UC mode (perhaps emanating from local tasks that only execute in the UC mode).
- HI-criticality packets that are relevant to UC mode; perhaps with an increased number of frames and/or alternative routes.
- LO-criticality packets that are relevant to UC mode; perhaps with an increased number of frames and/or alternative routes.
- UC packets that originate from other nodes and are being routed through this node.

All UC packets are transmitted with a priority higher than those used for the usual HI-criticality and LO-critically traffic. It is assumed that there is a finite number of packets to be communicated within the UC mode. Perhaps a single packet per originating task (i.e. these tasks are single-shot rather than recurrent). Analysis can again be used to determine how long it will take for such flows to reach their destinations when there are parts of the network unavailable and faults being experienced in the operational parts. Of course if the network is partitioned then it will not be possible to deliver the UC packets unless each partition has a relevant sink.

In the UC mode if there are currently no UC packets to transmit then other HI-criticality packets can be sent. They would have lower priority and hence would not interfere with newly arrived UC packets (from either the host node or being forwarded from other nodes); in general they would not however be guaranteed to arrive before their deadlines. It is assumed that LO-criticality packets, other than those promoted to UC, are not transmitted in the UC mode.

V. Evaluation

In this section, we consider the evaluation of the AirTight ultra-criticality mode change via simulation. The simulator is a discrete event simulation which allows analysis of the latencies of packet flows, and transmission of the mode change. It allows various faults to be defined with different probabilities and locations affected, and individual nodes to be disabled during simulation. The simulator also supports GUI visualisation of the network in the process of simulation, indicating the status of the nodes and their transmission buffers.

In order to evaluate the performance of the protocol, it is important to consider the length of time taken to deliver the UC mode change packet throughout the network, and the UC packets. In addition, we can also assess the impact upon the delivery rates and deadlines of the originally present HI-critical and LO-critical packets.

The case study described in Section II and our previous AirTight work [2] is used for the evaluation.

The slot table size used in this example case study is 30, which is equivalent to 5 copies of the 6-slot table used in [2]. The example topology is shown in Figure 1. The topology has been modified from that used in [2] by the addition of a number of additional links from nodes 9 to 4, 14 to 2, 19 to 1 and 24 to 3 (and since links are symmetric, the reverse). This provides additional redundancy which is required for providing fault tolerance in the event of a shaft break failure disconnecting the original primary wireless link.

The fault case selected for the experimental case study models a shaft break event occurring in the upper right section of the topology. Its effects upon the network are as illustrated in Figure 2. Nodes 8 and 6 fail permanently and the link from node 9 to node 3 is permanently disconnected. This loss of nodes fits with the requirements for the mode change in the aircraft case, in that a small proportion of the network nodes and connection links are lost as a result of the failure. The transmission of the UC mode change is initiated by node 5, which is informed of the shaft break by a reading from one of its directly connected sensors. Upon entering UC mode, nodes 5, 7, 10, 15 and 20 begin

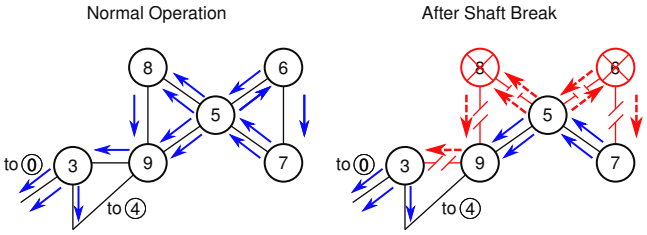


Fig. 2. Effects of the fault on the network (only the affected section shown)

executing a special processing task to gather logging data about the shaft break failure. When this task completes, these nodes transmit *UC traffic* – a data flow destined for node 0 for the central module to convey the logged data.

In an emergency situation, it is important for the mode change to be propagated across the network rapidly. The time taken from the mode change event occurring to the notification propagating across the network is considered in this section. For each case, two metrics are used: the time for the central node which has the wired link to the rest of the network (identified as node 0) to be notified, and the time for every node in the network to be notified. During the flood propagation, the network is subject to different levels of unrelated 'normal' faults, which manifest in a given probability of a transmission failing. This probability is uniform across the network, regardless of location. The length of these fault bursts is increased in the series of experiments performed.

The role of the simulator is to enable different scenarios and fault models to be explored. Clearly this is much easier to do with a simulator than a test-bed. One of the options available with the simulator is to either simulate worst-case fault behaviour, or to model fault arrivals via various stochastic processes. Another choice is whether to assume that each 'link' in the wireless network has dependent or independent faults. For dependent behaviour a fault hits all links at the same time - thus a routed message will perhaps only suffer interference from faults on one of its hops. With independent faults each hop could suffer this interference.

In the following examples of runs of the simulator we first force the faults to occur at the 'worst possible time' but assume faults are dependent. We then consider independent but stochastically modeled faults.

Figure 3 demonstrates the increasing time taken for the distribution of the mode change with increasing length of transient faults. In all experiments time is measured in numbers of slots.

For this experiment, the probability of transmitted data packets being interfered with during the fault interval is 100% - it is assumed, for a worst case, that the transient fault is completely destructive of ongoing traffic. Obviously, the time taken to inform the central node 0 is lower than the time taken to inform every node within the network. It is notable that as the fault length is increased, several discontinuities occur in which the elapsed time required to propagate the mode change increases suddenly. These occur as a result of the interaction between the periodic scheduling table of AirTight and the fault definition.

In this experiment the distribution queues which control the order for transmission of the mode change messages are setup to direct the mode change messages towards the central node 0. We found that this static property almost halved the delivery time by comparison with an arbitrary ordering of the distribution queues.

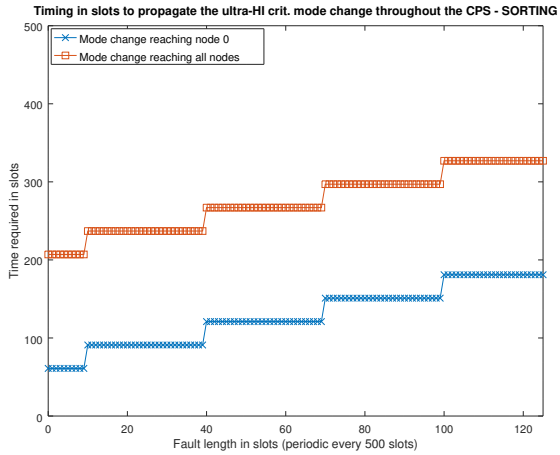


Fig. 3. Timing with ordered queues

Assuming the mode change in this case study represents an aircraft engine shaft break, a number of actions would be required in order to respond to the mode change. Firstly, we would assume that the network node detecting and signalling the mode change would have to perform some processing to determine the nature and effects of the fault that triggered the mode change. Then it would communicate with its coordinator node 0, transmitting some UC traffic in order to transmit additional data via a multi-hop route. Also, the other central nodes from each of the functional regions illustrated in Figure 1 (nodes 10, 15 and 20) would, on receiving the communicated mode change, determine the control effects which are necessary and then transmit the relevant data back to node 0. Given that these communication data flows could involve instructions to other engines that detail how they would have to respond to compensate, this would be transmitted at the highest priority after the mode change itself.

We now consider the latencies involved in the transmission of this ultra-criticality (UC) data traffic. Five UC data flows are activated, referred to as UC1 to UC5.

Figure 4 shows the latencies experienced for the 5 UC data flows, indicating the time following their injection into the network for the complete packets to reach their destination at node 0. It is assumed in Figure 4 that the processing delay to generate UC traffic is very short, effectively less than a single table, so the packets are ready to be transmitted and present in the buffer the next time the source node has a transmission slot.

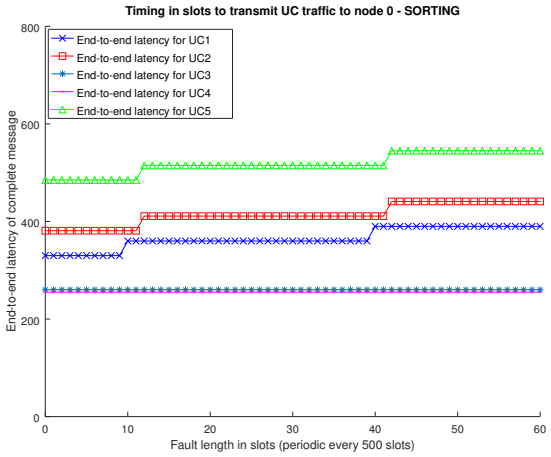


Fig. 4. Latencies for UC data transmission

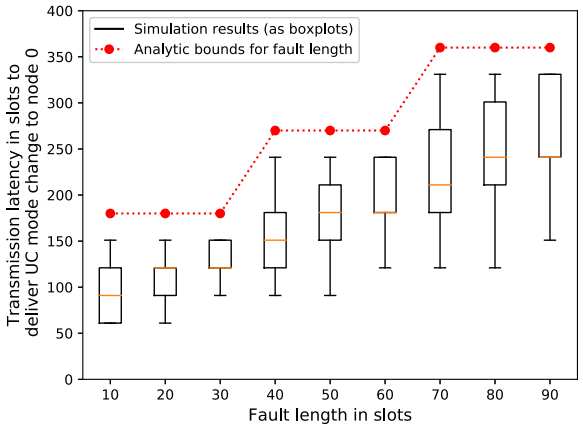


Fig. 5. Latencies for UC Mode Change Message

However, since UC flood packets (with the highest priority) are present in the network, the generated UC data will not be immediately transmitted.

The data series for the latency values is generated by varying the transient fault length which occurs in the network. These faults are assumed to begin immediately upon the mode change, i.e. upon the injection of the UC data traffic. The period of the faults is 500 slots, and the faults are assumed to recur at the beginning of this interval, network-wide (that is, all links in the network are affected simultaneously).

Since flows UC1 and UC2 share a majority of the same route, it is as expected that UC2 has a higher latency than UC1. UC3 and UC4 have disjoint routes, so they do not mutually interfere with each other, and achieve broadly the same latency even though they have different injection times. UC5 experiences the highest latencies since it may receive interference from UC1 and UC2 (due to their requirement to route via node 4).

We now show a result from simulating independent faults that arrive stochastically. Clearly many different arrival patterns can be experimented with. Faults hitting each link of a routed message are likely to be rare; but should nevertheless be investigated. In the following experiment each link experiences a fault that arrives randomly between the arrival of the UH mode change packet and that time plus the table length. So this packet can potentially propagate through the network without interference from faults but the probability of it suffering multiple faults is not negligible. Figure 5 shows a set of box plot results for increasing durations of faults. For each fault duration 1000 simulations were undertaken. Also shown on this figure is the analytical upper bound calculated using the analysis reported in previous work on AirTight [2]. Note, as expected, the longest propagation time observed in the simulation experiments is less than this bound.

VI. Conclusion

In a CPS system, a permanent fault may occur in such a way as to require a different protocol response from those normally assumed in the case of transient faults, such as retransmission and alternative routes. Specifically, it may require a functional mode change to be signalled throughout the network in order to inform the entire system of an emergency situation, as well as triggering the dropping of LO-criticality work. This paper has demonstrated the modification of the AirTight protocol in order to support these more challenging fault scenarios, with only a minor modification of the logic for data distribution at the highest priority level. The timing characteristics of this mode change data and associated logging traffic have been investigated via simulation to demonstrate its performance in the presence of a variety of fault intensities.

Acknowledgements

The research described in this paper is funded, in part, by the EPSRC grants MCCps (EP/P003664/1). No new primary data were created during this study.

References

[1] S.K. Baruah, A. Burns, and R. I. Davis. Response-time analysis for mixed criticality systems. In *Proc. IEEE Real-Time Systems Symposium (RTSS)*, pages 34–43, 2011.

[2] A. Burns, J. Harbin, L.S. Indrusiak, I. Bate, R.I. Davis, and D. Griffin. Airtight – a resilient wireless communication protocol for mixed-criticality systems. In *Proc. RTCSA*, 2018.

[3] S. Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *Proc. of the IEEE Real-Time Systems Symposium (RTSS)*, pages 239–243, 2007.

[4] X. Zhao, H. Gao, G. Zhang, B. Ayhan, F. Yan, C. Kwan, and J.L Rose. Active health monitoring of an aircraft wing with embedded piezoelectric sensor/actuator network: I. defect detection, localization and growth monitoring. *Smart Materials and Structures*, 16(4):1208, 2007.

Mixed Criticality Probabilistic Real-Time Systems Analysis using Discrete Time Markov Chain

Jasdeep Singh, Luca Santinelli, Guillaume Infantes, David Doose and Julien Brunel
ONERA -DTIS Toulouse, name.surname@onera.fr

***Abstract*—In this work, we demonstrate an effective way of applying formal methods to model and analyze mixed criticality probabilistic real-time systems. By probabilistic real-time system, we mean a real-time system in which the execution times of the task jobs are described with probabilistic worst-case execution time distributions. The criticality modes (high or low criticality) are defined as events with the associated probability of occurrence. The probabilistic response time is used to characterize the criticality mode of the jobs. Discrete Time Markov Chain is built to represent job criticality modes and mode combinations. We apply them to formally characterize and study the system criticality behaviour at runtime as well as the probability of the system entering high criticality mode. We also propose a strategy to perform a resource efficient selective dropping of low criticality jobs to improve the deadline miss probability of high criticality jobs. This strategy is an improvement on classical case of dropping all the low criticality jobs.**

***Index Terms*—Probabilistic real-time system, Probabilistic schedulability analysis, Mixed criticality, Formal methods, Discrete Time Markov Chain.**

I. Introduction

An increasingly important trend in developing real-time systems is the integration of applications with different levels of criticality. In a Mixed Criticality (MC) real-time system there are tasks with two or more distinct criticality levels e.g., safety critical – HI-criticality, mission critical or low critical – LO-criticality.

MC systems are defined to execute in a number of criticality modes, each specifying a particular execution condition and criticality. All the possible modes have to be characterized and analyzed in order to guarantee the predictability of the system; [4] presents a complete overview of the MC problem.

The gap between the actual execution behaviour and the worst-case bounds may be significantly large. The MC scheduling allows for less important tasks to execute in these gaps under normal circumstances, and may be dropped in an occasional situation where jobs of higher importance execute beyond their estimated execution time.

Another parallel trend in the real-time systems is to use probabilistic approaches. Probabilistic representations attempt to describe the (epistemic) uncertainty within a system with random variables [7]. In particular, the notion of probabilistic Worst-Case Execution Time (pWCET) is currently emerging [6], [7]. This is motivated by traditional rigorous and deterministic WCET analysis which may lead to pessimism, since the occurrence of WCET happens under highly pathological and extremely unlikely circumstances. The pWCET is a probability distribution that upper bounds any task execution time under every possible execution conditions [7]. In it, there are multiple WCET thresholds and the probabilities associated to them.

A real-time system with probabilistic parameters described with random variables e.g. pWCETs, is called probabilistic real-time system (pRTS). Characterizing certain pRTSs as hard real-time or soft real-time is left to the discretion of the developer because it depends on the required safety thresholds of the application.

As demonstrated by [12], [18], MC problems can be approached with probabilities to quantify and manage the unlikely events such as high criticality modes. It is our belief that a tighter coupling between MC and probabilistic frameworks can end up into 'smart' schedulers for more efficient utilization of the computational resource. This is because both, probabilistic approaches and the mixed criticality models, are motivated by safely minimizing the unused resources without sabotaging the real-time system's functioning. Obtaining a feasible analysis of such a complex problem is possible using formal methods. The formal MC probabilistic system representation can be subject to rigorous model checking. The validated probabilities extracted can be safely leveraged into scheduling decisions.

Contributions. This paper is about describing and studying the execution of MC jobs in pRTSs with the use of formal methods. We distinguish and approach two problems. The first problem is: **P1 – to quantify the probability for the system to execute in high criticality mode**. We propose a formal model with associated model checking to extract the probability of the system taking paths through high criticality modes. Formal methods are used to ensure that both the MC models and the MC analysis with probabilities are correct (validated).

The second problem is: **P2 – to control the schedulability of HI-criticality jobs based on their deadline miss probability**. Here, we reason a strategy to avoid dropping all the LO-criticality jobs in the system, once the system is in high criticality mode. The analysis which we develop selectively decides which LO-criticality job to drop based on its effects on HI-criticality jobs. With that, we are able to increase the number of LO-criticality jobs which can execute with HI-criticality jobs without jeopardizing their timing constraints.

Organization. Section II details the background of this paper in terms of notations and preliminaries. Section III presents the MC modeling and the formal methods used to model jobs behaviour and criticality modes with probabilities. Section IV illustrates the analysis we propose to calculate the probability

of system entering high criticality mode configurations. Section VI is for concluding remarks and future work.

A. Related work

In recent years, many attempts have been made in introducing probability to the modeling and schedulability analysis of real-time systems.

Safety-critical applications have to account for the worst-case behaviours that can possibly happen. The 'best' modeling of task parameters has to assure the coverage of any of the execution conditions, worst-cases included [12]. LO-critical applications rely on less constrained/demanding models. The guarantees of their models are not as strict as those for safety critical applications [23], [9].

Based upon pWCETs, execution time distributions, and/or other possible parameters, probabilistic schedulability analyses have been developed. They compute the probability of missing a deadline and check if it is small enough for safety requirements. Tia et al. [22] focus on unbalanced heavy loaded system, with maximum utilization larger than 1 and much smaller average utilization, and provide two methods for probabilistic schedulability guarantees. Lehoczky [14] proposes the first schedulability analysis of task systems with probabilistic execution times. This work is further extended to specific schedulers, such as Earliest Deadline First (EDF) in [24] and under fixed priority policy in [10]. Statistical response-time analysis, e.g., [16], can be further done to real-time embedded systems based upon those probabilistic schedulability analysis. Some works exist that apply discrete pWCET distribution to schedulability analysis [8], [17], [21]. They rely on the convolution operation between distributions to find tasks discrete probabilistic response time.

In the real-time systems community, some existing work in MC scheduling considers the concept of graceful degradation and proposes effective scheduling techniques, such as fluid-based scheduling [2], utilization-based earliest deadline first with virtual deadlines (EDF-VD) [15], putting restriction on the asymptotic rate-based correctness notation [20], [1], and involving mixed-criticality weakly hard constraints [11]. [18] applies probability thresholds into MC schedulability conditions; different schedulability conditions are defined from LO-criticality to HI-criticality. [12], [18] represent the first works to make use of probabilities for scheduling decisions.

II. Notations and Preliminaries

Here are some notions we make use of in this work.

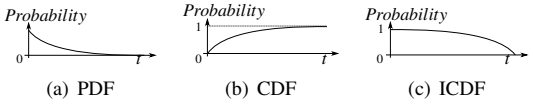
A. Probability fundamentals

Given a continuous random variable $\mathcal{X}$, the Probability Density Function (PDF) $f_{\mathcal{X}}(x)$ of $\mathcal{X}$ gives the probability that a value extracted from $\mathcal{X}$ lies between a and b as: $Pr(a \leq \mathcal{X} \leq b) \stackrel{def}{=} \int_a^b f_{\mathcal{X}}(x)dx$. In $[0,+\infty)$, $\int_0^\infty f_{\mathcal{X}}(x)dx = 1$. In case of $\mathcal{X}$ being a discrete random variable, the PDF gives the probability of an event x, $f_{\mathcal{X}}(x) \stackrel{def}{=} Pr(\mathcal{X} = x)$.

The Cumulative Distribution Function (CDF) is represented as $F_{\mathcal{X}}(x)$ and the Inverse Cumulative Distribution Function (ICDF) as $F'_{\mathcal{X}}(x)$.

The convolution of two continuous PDFs $f_{\mathcal{X}}(x)$ and $g_{\mathcal{Y}}(y)$, denoted by $\otimes$, refers to the summation of the random variables $\mathcal{X}$ and $\mathcal{Y}$, given as: $f \otimes g(z) = \int_{-\infty}^{\infty} f(z)g(x-z)dz$. The convolution of more than two PDFs is represented as $\bigotimes_i \mathcal{C}_i$. In our case, the random variable $\mathcal{X}$ represents the duration of execution of a task or job.

Figure 1 illustrates an example of a random variable represented respectively with the PDF, the CDF and the ICDF.



(a) PDF (b) CDF (c) ICDF

Fig. 1: PDF, CDF, and ICDF representations of a generic random variable.

B. Probabilistic computational models

A periodic real-time application Γ has m tasks $\Gamma = \{\tau_1, \tau_2, \ldots \tau_m\}$, $m \in \mathbb{N}^+$ where $\mathbb{N}^+$ is the set of positive natural numbers. A task τ_i is a tuple $\tau_i = (\mathcal{C}_i, T_i, D_i)$ where $\mathcal{C}_i$ is the pWCET PDF, T_i is the period, D_i is the deadline of the task; $D_i \leq T_i$. Both T_i and D_i are deterministic parameters – single valued. $\mathcal{C}_i$ can be continuous or discrete worst-case distribution.

Since the tasks are periodic, each j-th instance of a task τ_i is a job J_{ij}. The job model is such that: $J_{ij} = (\mathcal{C}_i, a_{ij}, d_{ij}, p_{ij})$, $j = 1,2,3....$ J_{ij} arrives at time $a_{ij} = (j-1) \cdot T_i$; d_{ij} is the job absolute deadline, $d_{ij} = a_{ij} + D_i$; p_{ij} is the job priority, 0 being the highest priority i.e. if $p_{ij} < p_{kr}$ then J_{ij} has higher priority than J_{kr}. The hyperperiod, defined as the least common multiple of the periods of all the tasks periods $lcm(T_i)$, $i = 1,2,...,m$ is the scope of our analysis. In the hyperperiod there are n jobs, with n_i jobs for each task τ_i, $n = \sum_{k=0}^{m} n_k$. The real-time application can be represented with the n jobs, $\Gamma = \{J_{ij}\}$ $i = 1,2,...,m; j = 1,2,...,n$.

In pRTSs, the worst-case response time of a job is a random variable, represented as $\mathcal{RT}$.

Definition 1 (Job probabilistic worst-case response time)**.** *A probabilistic worst-case response time pWCRT for a job J_{ij}, denoted as $\mathcal{RT}_{ij}$, is a probabilistic distribution which upper bounds every possible job response times due to concurrent job interferences and preemptions during its execution. It is represented with PDF as $f_{\mathcal{RT}_{ij}}(x)$, CDF as $F_{\mathcal{RT}_{ij}}(x)$ and ICDF as $F'_{\mathcal{RT}_{ij}}(x)$.*

Definition 2 (Job deadline miss probability)**.** *The probability that a job executes until a time greater than or equal to the respective deadline is called deadline miss probability of the job. For a job J_{ij}, it is denoted as P_{ij}^{dm}:*

$$P_{ij}^{dm} = Pr(J_{ij} \text{ finishes executing after } d_{ij}) \stackrel{def}{=} \int_{d_{ij}}^{\infty} f_{\mathcal{RT}_{ij}} dx. \quad (1)$$

Assumptions: In a probabilistic framework, the schedulability analysis refers to the process to obtain the pWCRTs and deadline miss probabilities of the constituent jobs: a task set is schedulable if the deadline miss probability of each of its tasks is smaller than a certain required probability. The pWCRT

can be a discrete or a continuous probability distribution. The framework that we propose utilizes the given pWCRT of each job in the system. This framework is valid irrespective of whether the input (pWCET) or the output (pWCRT) distributions of the analysis are discrete or continuous. Thanks to the worst-case assumption of task execution scenarios, pWCET of the tasks are independent [5]. All the execution interferences during the job executions are considered in the scheduling analysis and are represented by their pWCRT. It is the probabilistic counterpart of the same assumption in the deterministic context. In this work, the scheduling follows the EDF preemptive policy which defines the job ordering by static job-wise priorities p_{ij}. It applies to single-core platforms, and we assume the tasks' jobs are suspended at the deadline.

III. Mixed criticality Problem

In the two-criticality-level case, each job is designated as being of either higher criticality HI-criticality or lower criticality LO-criticality. The HI-criticality mode is where the job executes in highly critical (and more demanding) conditions – critical function or fault recovery; a LO-criticality mode is the nominal working condition for the job where it executes in normal conditions. Having a higher criticality is regarded as giving more execution time to the task.

A HI-criticality job J_{ij}^{HI} is the tuple: $J_{ij}^{\text{HI}} \stackrel{def}{=} (C_i, a_{ij}, d_{ij}, p_{ij}, l_{ij}, \chi_{ij})$. C_i, a_{ij}, p_{ij} and d_{ij} are as defined earlier. χ_{ij} is the job criticality level defined for a job J_{ij} [3] which can take two values at runtime: HI and LO; $\chi_{ij} = \{\text{HI}, \text{LO}\}$. $l_{ij} \leq D_i$ describes the threshold with which we define the job criticality mode.

A LO-criticality job J_{kr}^{LO} is the tuple: $J_{kr}^{\text{LO}} \stackrel{def}{=} (C_k, a_{kr}, d_{kr}, p_{kr}, \chi_{kr})$. χ_{kr} for a LO-criticality job can take only one value, $\chi_{kr} = \{\text{LO}\}$.

For the jobs, the criticality level of its task is inherited e.g., for J_{ij} and J_{ik} have the same criticality level as of the task τ_i level. However, the actual criticality mode of the jobs can change at runtime depending on their scheduling, in turn affecting the system criticality mode.

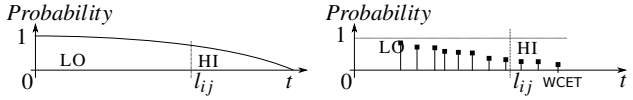
The real-time application is formed from these tasks which is partitioned between their HI-criticality jobs and LO-criticality jobs. $\Gamma^{\text{HI}} = \{J_{ij}^{\text{HI}}\}$ is the set of high criticality jobs with n^{HI} number of HI-criticality jobs; $\Gamma^{\text{LO}} = \{J_{kr}^{\text{LO}}\}$ is the set of LO-criticality jobs with n^{LO} the number of LO-criticality jobs; $\Gamma = \Gamma^{\text{HI}} \cup \Gamma^{\text{LO}}$ and n is the total number of jobs in the hyperperiod, $n = n^{\text{HI}} + n^{\text{LO}}$. With tasks, it is m^{HI} the number of HI-criticality tasks, and m^{LO} the number of LO-criticality tasks; $m^{\text{HI}} + m^{\text{LO}} = m$.

Classically, the definition of system criticality is such that: the system enters high criticality mode whenever at least one of the HI-criticality job enters high criticality mode [3]. We propose the following more generic and flexible definition of system criticality.

Definition 3 ((k,n) System criticality)**.** *The system criticality level χ is high (HI) if at least k^{HI} out of n^{HI} HI-criticality jobs enter high criticality mode.*

Using the above definition for system criticality allows the flexibility to choose the value of k^{HI} depending on the system. This also implies that a k^{HI} greater than 1 is less pessimistic than the classical definition of system criticality. Because of the nature of the pRTSs, the event of the system entering high criticality mode is not deterministically known anymore: there exists a probability of the system entering the high criticality mode. This is why we need to use reliable probabilistic analysis tools to analyze such a system.

Criticality threshold: We define the job criticality mode using a threshold $l_{ij} \leq D_i$ which applies to the job pWCRT. As shown in Figure 2(a) with HI- and LO-criticality regions, if the job finishing time is in $[l_{ij}, \infty)$, the job is considered to execute in high criticality mode, otherwise the job executes in low criticality mode, $[0, l_{ij})$.



(a) Continuous (b) Discrete

Fig. 2: pWCRT of job J_{ij} in its ICDF in (a) continuous and (b) discrete form . High and low criticality regions are separated by l_{ij}. Low and High criticality zones denoted as LO and HI.

The probability P_{ij}^{HI} that a job J_{ij} executes in the high criticality mode is:

$$P_{ij}^{\text{HI}} \stackrel{def}{=} \int_{l_{ij}}^{\infty} f_{\mathcal{RT}_{ij}}(x)dx. \tag{2}$$

If the pWCRT is discrete as shown in Figure 2(b) with WCET as the maximum possible execution time, $P_{ij}^{\text{HI}} \stackrel{def}{=} \sum_{x \geq l_{ij}}^{WCET} f_{\mathcal{RT}_{ij}}(x)$.

The MC definitions and modelling we propose, is slightly different than the classical ones with multiple WCET thresholds [23], [12]. With the MC modeling via pWCRT, it is possible to distinguish the job behaviour at runtime, which, otherwise impossible to do with pWCETs and WCET thresholds. This allows to relate the system criticality to the actual job execution which includes the job waiting time due to preemptions and postponements. Nonetheless, the MC analysis we propose is general enough to apply to WCET thresholds also. Note that in the latter case, every job of the same task would have the same criticality mode.

Discrete Time Markov Chain: We assume that a task set is given, it is scheduled using EDF scheduling policy, and the pWCRT for each job in the hyperperiod is known. The system criticality modes are modelled as a Discrete Time Markov Chain (DTMC). The choice of DTMC makes it possible to simply arrange the readily available probabilities from the schedulability analysis. System depiction as states replicates the switching the real-time system between high the low criticality modes. The transitions between those states can be labelled with the probability of it being chosen. Moreover, DTMC allows modelling of a probabilistically distributed system subject to its mathematical foundation. This is unlike a linear system, like Petri net or automata, where discrete actions form a complex explorable tree. DTMC is subject to formal model checking in which path properties or the probability of reaching certain states can be formally checked. As we will

see, this is useful to know the probability of a path taken by the system. A DTMC M is defined as a set of states S and state transitions given by a Q-matrix Q, $M = (S, Q)$ [19]. In the following, we give the basics to build DTMC for mode changes in MC pRTSs and the probabilistic properties which are formally verified with PRISM Model Checker [13]. It should be noted that property verification for a probabilistic system returns a probability of that property being true.

System criticality mode modeling: We present here the DTMC model for problem **P1**, $M^{crit} = (S^{crit}, Q^{crit})$;

To build M^{crit} it is required to consider only the HI-criticality jobs in this DTMC. It is because only HI-criticality jobs contribute to decide the system criticality. The contribution of the LO-criticality jobs is included in the pWCRTs of the HI-criticality jobs. For each job $J_{ij}^{\text{HI}} \in \Gamma^{\text{HI}}$, the set of states $S_{ij} = \{J_{ij}^{\text{HI}}, HC_{ij}, LC_{ij}\}$ is defined. State HC_{ij} represents execution of J_{ij}^{HI} in high criticality mode ($[l_{ij}, \infty)$) and state LC_{ij} represents execution of J_{ij}^{HI} in low criticality mode ($[0, l_{ij})$). The state J_{ij}^{HI} is simply a passing state used for the ease of modeling and representation; it has no contribution to the analysis. The set of states S^{crit} for the whole system is defined as an union of the sets S_{ij}, $\forall i, j$ such that job J_{ij} is a HI-criticality job: $S^{crit} \stackrel{def}{=} (\bigcup_{i,j} S_{ij}) : J_{ij} \in \Gamma^{\text{HI}}$.

S^{crit} is ordered in the increasing priorities of the jobs that it contains, for the ease of formalization and presentation. Any other ordering would be possible, since the DTMC does not represent the scheduling but only the criticality configurations. The set of states and transitions in M^{crit} is shown in Figure 3 for an example task set Γ. There are unidirectional transitions $J_{ij}^{\text{HI}} -> HC_{ij}$, $J_{ij}^{\text{HI}} -> LC_{ij}$, $HC_{ij} -> J_{ab}$, $LC_{ij} -> J_{ab}$; such that $p_{ab} > p_{ij}$ and the priority difference $|p_{ij} - p_{ab}|$ is minimum $\forall i, j, a, b, J_{ab}, J_{ij} \in \Gamma^{\text{HI}}$. The initial state is a J_{ab} such that p_{ab} is minimum and $J_{ab} \in \Gamma^{\text{HI}}$.

The state transitions are labelled such that the sum of the probabilities of all the outgoing transitions is equal to one. Each transition emanating from a state J_{ij} to HC_{ij} is labelled with the probability P_{ij}^{HI} from Equation (2). This implies, each transition emanating from a state J_{ij} to LC_{ij} is labelled with the probability $1 - P_{ij}^{\text{HI}}$. The transition matrix Q^{crit} defined as:

$$
\begin{array}{c}
\begin{array}{c|ccccccc}
 & J_{11} & HC_{11} & LC_{11} & \dots & J_{m^{\text{HI}}n^{\text{HI}}} & HC_{m^{\text{HI}}n^{\text{HI}}} & LC_{m^{\text{HI}}n^{\text{HI}}} \\
\hline
J_{11} & 0 & P_{11} & 1-P_{11} & \dots & 0 & 0 & 0 \\
HC_{11} & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\
LC_{11} & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\
J_{12} & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\
HC_{12} & 0 & 0 & 0 & \dots & 1 & 0 & 0 \\
LC_{12} & 0 & 0 & 0 & \dots & 1 & 0 & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
J_{m^{\text{HI}}n^{\text{HI}}} & 0 & 0 & 0 & \dots & 0 & P_{m^{\text{HI}}n^{\text{HI}}} & 1-P_{m^{\text{HI}}n^{\text{HI}}} \\
HC_{m^{\text{HI}}n^{\text{HI}}} & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\
LC_{m^{\text{HI}}n^{\text{HI}}} & 0 & 0 & 0 & \dots & 0 & 0 & 0
\end{array}
\end{array}
\tag{3}
$$

To refer to an element of the matrix, $Q^{crit}(State_r, State_c)$ gives the probability of transition from a state $State_r$ in the row to a state $State_c$ in the column, e.g. probability of transition from J_{11} to HC_{11} is referred as $Q^{crit}(J_{11}, HC_{11})$, equal to P_{11}.

As we see, the DTMC models construction is quite straightforward without involving any mathematical operations like convolution or upper-bounding of any kind. It is simply arranging the probabilities obtained from the schedulability analysis into formally verifiable DTMC structure. Thus, the safety of this construction comes the safety of the given schedulability analysis used to obtain the pWCRTs.

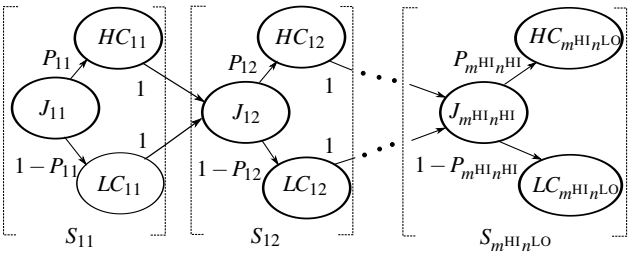


Fig. 3: System DTMC model M^{crit}, assuming $J_{11}, J_{12}, \dots, J_{m^{\text{HI}}n^{\text{HI}}} \in \Gamma^{\text{HI}}$ and such that $p_{11} \leq p_{12} \leq \dots \leq p_{m^{\text{HI}}n^{\text{HI}}}$.

IV. Criticality analysis

In this section, we detail our proposition to tackle the problem **P1**. With DTMC $M^{crit} = (S^{crit}, Q^{crit})$ Matrix (3), we define the meaning of system criticality from the criticality levels of the jobs. Then, we quantify the probability of the system criticality by exploring M^{crit} with formal model checking. We name this analysis 'criticality analysis' (*crit*). The definition of system criticality level translates into paths within the DTMC taken by the system through certain states. Each path has a probability of being taken at runtime. This is a probability for the system entering the high criticality mode.

A path D in M^{crit} represents the trace of the jobs taking high or low criticality modes at runtime. It is an ordered set of states $D \stackrel{def}{=} [State_1, State_2, ...State_{n^{\text{HI}}}]$, such that $Q^{crit}(State_k, State_{k+1}) > 0$ i.e. there exists a transition between two consecutive states. The probability of occurrence of the path D is $Pr(D)$, and it is computed by performing model checking on DTMC using the property: 'the maximum probability that the next state is $State_1$ AND the next to next state is $State_2$ AND the next to next to next...'. This property is formally written as: $Pmax = ?[X state = State_1 \ \& \ XXX state = State_2 \ \& XXXXX state = State_3 \dots]$. It should be noted that for each state, the next (X) is considered one from the initial state. Doing so defines a model checking property which navigates through the states.

For M^{crit}, there are $2^{n^{\text{HI}}}$ possible paths which start from the initial state. Examples of a path are: $D_1 = [J_{11}, HC_{11}, J_{12}, LC_{12} \dots, LC_{m^{\text{HI}}n^{\text{HI}}}]$, $D_2 = [J_{11}, LC_{11}, J_{12}, HC_{12}, \dots, HC_{m^{\text{HI}}n^{\text{HI}}}]$.

(k,n) system criticality: Here, we recall the Definition 3 that system is said to be in high criticality mode if at least k^{HI} out of n^{HI} jobs enter high criticality mode. The system enters high criticality mode in paths in which there is more than or equal to k^{HI} high criticality states HC. Such paths are denoted by the superscript kn.

The q-th path D_q^{kn} has a probability of occurrence $Pr(D_q^{kn})$. There are q_k paths which pass through a minimum of k_{HI} high criticality state. The exact value of q_k follows the mathematics of partitioning of numbers, which is left for future discussions. Now, using Definition 3 system enters the critical region if, D_1^{kn} occurred OR D_2^{kn} occurred OR $\dots$ $D_{q_k}^{kn}$ occurred. Thus, the probability P^{kn} that the system enters high criticality region is:

$$
P^{kn} \stackrel{def}{=} Pr(D_1^{kn}) + \dots + Pr(D_{q_k}^{kn}) = \sum_{q=0}^{q_k} Pr(D_q^{kn}). \tag{4}
$$

	C_i	$T_i = D_i$	χ_i
τ_1	$EXP(12)$	3	{HI,LO}
τ_2	$EXP(13)$	5	{HI,LO}
τ_3	$EXP(15)$	6	LO
τ_4	$EXP(18)$	10	LO
τ_5	$EXP(15)$	10	LO
τ_6	$EXP(16)$	15	LO

Fig. 4: Task set Γ parameters.

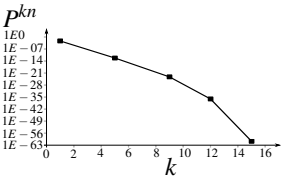


Fig. 5: P^{kn} from (k,n) system criticality.

As special cases, (1,n) and (n,n) system criticality are the two extremes of the (k,n) definition. (1,n) is to say that the system is in high criticality mode if at least one HI-criticality job is in high criticality mode: $P^{1n} \stackrel{def}{=} Pr(D_1^{1n}) + \ldots + Pr(D_{2^n-1}^{1n}) = \sum_{q=1}^{2^n-1} Pr(D_q^{1n})$. Where D^{1n} are the paths taken through at least one high criticality states. (n,n) is to say that the system criticality level as high whenever all the HI-criticality jobs are in high criticality mode: $P^{nn} \stackrel{def}{=} Pr(D_1^{nn})$. Where D_1^{nn} is the path taken through all the high criticality states. This concludes the solution to problem **P1**.

Example 1. *For this and the next section, we assume a task set* Γ *as shown in Table 4. All the timing parameters are in unit time and the* C_i *are exponential distributions with rate RATE, represented as* $EXP(RATE)$. Γ *is composed of* 6 *tasks with a total of* 31 *jobs. Tasks* τ_1 *and* τ_2 *are* HI*-criticality tasks, and tasks* τ_3, τ_4, τ_5 *and* τ_6 *are* LO*-criticality tasks. The threshold* $l_i = 0.7D_i$. Γ *is scheduled using preemptive EDF, the jobs are aborted at the deadline. The hyperperiod of this task set is* 30 *time units; in it, there are a total of* 16 HI*-criticality jobs and* 13 LO*-criticality jobs executing. The task set* Γ *from Example 1 is applied to this methodology for different* k_{HI}.

Figure 6 shows the value of P^{kn} *computed with Equation (4) as the value of k changes from the* (k,n) *system criticality definition Definition 3 . The probability that system enters* HI*-mode: using* ***(1,n)*** *is* $6.45E-03$ *and using* ***(n,n)*** *is* $2.70E-62$.

V. Deadline Miss Analysis

This section focuses on the deadline miss probability of HI-criticality jobs. The problem **P2** can be stated as: **the deadline miss probability of a job** J_{ij} **should be less than or equal to a given probability** $P^{dm,max}$. Here $P^{dm,max}$ is assumed to be given and is the requirement to meet in order to guarantee the probabilistic schedulability. Here, the focus is on the job to reduce its probability of deadline miss 2 . Solving **P2** is to define a MC scheduling algorithm that is able to improve the probability of deadline miss of jobs of choice.

Usually, MC scheduling directs that when the system is in high criticality mode, all the LO-criticality jobs are dropped to ensure the timing requirements of the remaining HI-criticality jobs [3]. Instead, the scheduling algorithms we propose acts by selectively dropping LO-criticality jobs whenever the deadline miss probability constraint is not met. It is conceived to minimize the number of LO-criticality jobs to drop allowing some of the LO-criticality jobs executing with HI-criticality jobs. This way, the computational resources are better used and the scheduling of HI-criticality tasks is not jeopardized. We present one such offline method is presented below.

Job strategy: The job strategy is the scheduling algorithm we propose to reduce the probability of deadline miss of a HI-criticality job without dropping all the LO-criticality jobs in the system. Classically, all the LO-criticality jobs are dropped in the high criticality mode. A choice to drop a single job in the ordered list of jobs requires complete re-evaluation of the whole system to prove optimality. This is because the response times depends on the execution of the previously executed jobs. The strategy to choose needs to ensure the optimality as well as the safety of the resulting schedule. The complexity of doing so for every job to prove optimality and safety is $O(n^n)$, where there are n jobs in the hyperperiod. Such complexity does not include the complexity of the probabilistic schedulability analysis applied; it is only for exploring all the jobs.

What we present here is not an optimal strategy to maximize the deadline miss probability reduction per job dropped. However, it is better than classical MC scheduling strategies in which all the LO-criticality jobs are dropped whenever the system enters high criticality mode [3].

Interference isolation: The set of LO-criticality jobs $\overline{J}^{over}(J_{ij}^{\text{HI}})$ which overlap to the execution of a job J_{ij}^{HI} is:

$$\overline{J}^{over}(J_{ij}^{\text{HI}}) \stackrel{def}{=} \{J_{gh} : p_{gh} < p_{ij}, d_{gh} > a_{ij}, J_{gh} \in \Gamma^{\text{LO}}\}. \quad (5)$$

The jobs in this set directly impose a probabilistic delay/backlog in the execution of J_{ij}^{HI}, as depicted in the Figure 6 by jobs J_{rt}^{LO} and J_{kj}^{LO}. All the jobs in $\overline{J}^{over}(J_{ij}^{\text{HI}})$ impose an indirect backlog to J_{ij}^{HI}. A certain amount of job backlog is passed in their order of priority and thus indirectly to the job J_{ij}^{HI}. The term 'interference isolation' refers to the separation of J_{ij}^{HI} from the indirect backlog.

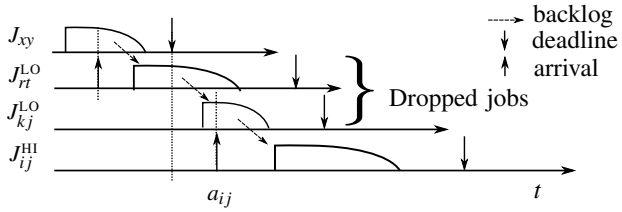


Fig. 6: The backlog to the HI-criticality job J_{ij}^{HI} reduces to zero by dropping jobs J_{rt}^{LO} and J_{kj}^{LO}.

Lemma 1 (Backlog isolation)**.** *The backlog for a* HI*-criticality job* J_{ij}^{HI} *reduces by the maximum amount if all the* LO*-criticality jobs in* $\overline{J}^{over}(J_{ij}^{\text{HI}})$ *from Equation (5) are dropped, given that the jobs are suspended at their respective deadlines.*

Proof. With continuous distributions defined in $[0,\infty)$, a job is always executing and always imparting a probabilistic backlog to next jobs until its deadline. For the job J_{ij}^{HI} this is when all these jobs are removed to ensure a maximum reduction in deadline miss probability. Due to the job suspension at the deadline, whenever there is minimum execution overlap, the backlog carried on to the next job reduces by a maximum. By maximum, it is meant the best possible effort to reduce the probability. The backlog to J_{ij}^{HI} from all the preceding jobs is minimized, once all the jobs in $\overline{J}^{over}(J_{ij}^{\text{HI}})$ are dropped. □

The job-level scheduling strategy we propose reduces the probability of deadline miss of a HI-criticality job by dropping all the LO-criticality jobs in $\overline{J}^{over}(J_{ij}^{\mathrm{HI}})$. Lemma 1 proves that dropping all those jobs ensures the maximum possible deadline miss probability reduction for J_{ij}^{HI}. Thus, the strategy is: 1) identifying $\overline{J}^{over}(J_{ij}^{\mathrm{HI}})$, 2) dropping all the jobs in $\overline{J}^{over}(J_{ij}^{\mathrm{HI}})$. Referring to the Figure 6, the backlog to the HI-criticality job J^{HI} minimizes by dropping jobs J_{rt}^{LO} and J_{kj}^{LO}: there is no effect from the job J_{xy} to the job J_{ij}^{HI} because J_{xy} suspends at the deadline. The HI-criticality job in observation still retains its own execution after dropping the jobs in the set $\overline{J}^{over}(J_{ij}^{\mathrm{HI}})$, that is after removing maximum interferences. The P_{ij}^{dm} obtained once dropping all the jobs in $\overline{J}^{over}(J_{ij}^{\mathrm{HI}})$, is the best (minimum) deadline miss probability we can achieve for J_{ij}^{HI}. If it is not enough to meet the constraint $P_{ij}^{dm,max}$, P_{ij}^{dm} is still larger than $P_{ij}^{dm,max}$, the problem for this job is unsolvable.

Example 2. *The task set Γ from Example 1 is analyzed. Figure 7 shows the probability of deadline miss P_{ij}^{dm} for* HI*-criticality jobs J_{26}, J_{14} and J_{24} when no* LO*-criticality job is dropped as $\{\emptyset\}$. Their corresponding set $\overline{J}^{over}(J_{ij}^{\mathrm{HI}})$ is shown and the jobs in it are dropped. The reduced value P_{ij}^{dm} for each job is also shown.*

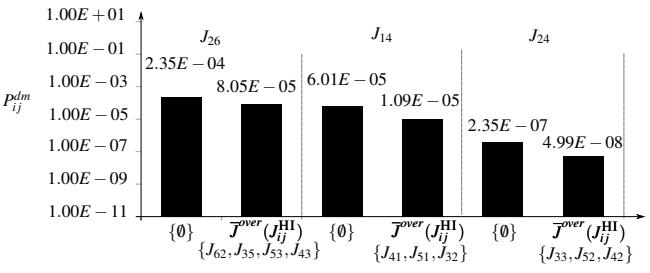


Fig. 7: Probability of deadline miss of the job J_{14} vs the LO-criticality jobs dropped.

VI. Conclusion and future work

With this work, we have proficiently applied probabilistic formal methods (DTMC) to model and analyze MC pRTSs. The pWCRT is used to define the MC behaviour of the jobs with a threshold parameter (l_{ij}) to distinguish between low criticality mode and high criticality mode for HI-criticality jobs. The DTMC representations of the runtime behaviour are studied to quantify the probability of the system entering HI-criticality mode. DTMC is also used to obtain deadline miss probability of HI-criticality jobs, and it can be easily extended to tasks in the future works. We have proposed heuristics to selectively drop LO-criticality jobs in order to achieve the required maximum probability of deadline miss. This study does not yet focus on the scalability. They are enhancements to existing MC schedulability that allow for a better use of the computational resource by reducing the number of dropped LO-criticality jobs.

References

[1] S. Baruah. A scheduling model inspired by control theory. In *Proceedings of the 6th International Real-Time Scheduling Open Problems Seminar*, 2015.

[2] S. Baruah, A. Burns, and Z. Guo. Scheduling mixed-criticality systems to guarantee some service under all non-erroneous behaviors. In *Proceedings of the 28th Euromicro Conference on Real-Time Systems*, 2016.

[3] S. K. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, S. van der Ster, and L. Stougie. Preemptive uniprocessor scheduling of mixed-criticality sporadic task systems. *J. ACM*, 62(2), 2015.

[4] A. Burns and R. Davis. Mixed-criticality systems: A review. http://www-users.cs.york.ac.uk/ burns/review.pdf+, 2017.

[5] L. Cucu-Grosjean. Independence - a misunderstood property of and for (probabilistic) real-time systems. Invited paper to the 60th birthday of A. Burns, 2013.

[6] L. Cucu-Grosjean, A. G. Gogonel, and M. Dorin. Probabilistic real-time scheduling. In M. Chetto, editor, *Reat-time Systems Scheduling*. ISTE/Wiley, 2014.

[7] R. I. Davis, A. Burns, and D. Griffin. On the meaning of pwcet distributions and their use in schedulability analysis. In *In Proceedings Real-Time Scheduling Open Problems Seminar at ECRTS*, 2017.

[8] J. L. Diaz, D. F. García, K. Kim, C.-G. Lee, L. L. Bello, J. M. López, S. L. Min, and O. Mirabella. Stochastic analysis of periodic real-time systems. In *RTSS: Proceedings of the 23rd IEEE Real-Time Systems Symposium*, page 289. IEEE Computer Society, 2002.

[9] R. Ernst, A. Burns, L. Thiele, and J. L. Rhun. Mixed critical system design and analysis. In *Proceedings of the 12th International Conference on Embedded Software, EMSOFT*, 2012.

[10] M. Gardner and J. Liu. Analyzing stochastic fixed-priority real-time systems. In *the 5th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 1999.

[11] O. Gettings, S. Quinton, and R. I. Davis. Mixed criticality systems with weakly-hard constraints. In *Proceedings of the 23rd International Conference on Real Time and Networks Systems*, 2015.

[12] Z. Guo, L. Santinelli, and K. Yang. Edf schedulability analysis on mixed-criticality systems with permitted failure probability. In *21th IEEE International Conference on Embedded and Real-Time Computing System and Applications*, 2015.

[13] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In G. Gopalakrishnan and S. Qadeer, editors, *Proc. 23rd International Conference on Computer Aided Verification CAV*, volume 6806 of *LNCS*. Springer, 2011.

[14] J. Lehoczky. Real-time queueing theory. In *the 17th IEEE Real-Time Systems Symposium (RTSS)*, 1996.

[15] D. Liu, J. Spasic, N. Guan, G. Chen, S. Liu, T. Stefanov, and W. Yi. EDF-VD Scheduling of Mixed-Criticality Systems with Degraded Quality Guarantees. In *Proceedings of the 37th IEEE Real-Time Systems Symposium*, 2016.

[16] Y. Lu, T. Nolte, I. Bate, and L. Cucu-Grosjean. A statistical response-time analysis of real-time embedded systems. In *the 33rd IEEE Real-Time Systems Symposium (RTSS)*, 2012.

[17] D. Maxim and L. Cucu-Grosjean. Response time analysis for fixed-priority tasks with multiple probabilistic parameters. In *the 34th IEEE Real-Time Systems Symposium (RTSS)*, 2013.

[18] D. Maxim, R. I. Davis, L. Cucu-Grosjean, and A. Easwaran. Probabilistic analysis for mixed criticality systems using fixed priority preemptive scheduling. In *Proceedings of the 25th International Conference on Real-Time Networks and Systems, RTNS*, 2017.

[19] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., 1st edition, 1994.

[20] I. Saha, S. Baruah, and R. Majumdar. Dynamic scheduling for networked control systems. In *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control, HSCC*. ACM Press, 2015.

[21] L. Santinelli and L. Cucu-Grosjean. A probabilistic calculus for probabilistic real-time systems. *ACM Trans. Embedded Comput. Syst.*, 14(3), 2015.

[22] T. Tia, Z. Deng, M. Storch, J. Sun, L. Wu, and J. Liu. Probabilistic performance guarantee for real-time tasks with varying computation times. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 1995.

[23] S. Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *Proceedings of the 28th IEEE International Real-Time Systems Symposium (RTSS)*, pages 239–243. IEEE Computer Society, 2007.

[24] H. Zhu, J. Hansen, J. Lehoczky, and R. Rajkumar. Optimal partitioning for quantized EDF scheduling. In *the 23rd IEEE Real-Time Systems Symposium (RTSS)*, 2002.

Supporting Graceful Degradation through Elasticity in Mixed-Criticality Federated Scheduling

Chris Gill, James Orr, and Steven Harris
Department of Computer Science and Engineering, Washington University in St. Louis, MO, USA
{cdgill, james.orr, sharris22}@wustl.edu

***Abstract*—Mixed-criticality federated scheduling (MCFS) lets parallel real-time tasks with different criticality levels co-exist on a common multi-core host, guaranteeing that lower-criticality tasks do not interfere with timely completion of higher-criticality tasks, even if tasks exceed their nominal utilizations. However, lower-criticality tasks may be suspended or dropped in MCFS, to accommodate any higher-criticality task's increased utilization requirements. Recent advances in elastic scheduling theory and concurrency platforms for parallel real-time (PRT) tasks offer a more graceful way to handle such resource reallocation, however, by exploiting *elasticity* of lower-criticality tasks' utilizations. In this paper we summarize those advances, provide a combined elastic multi-criticality PRT task model, sketch strategies for leveraging elasticity, and give brief illustrative examples showing how such elasticity can provide a more graceful means of resource reallocation in mixed-criticality federated scheduling.**

Keywords—mixed-criticality, parallel real-time systems, elastic scheduling, federated scheduling

I. Introduction

Parallel real-time computing has emerged as an important means to harness the increasing computational capacity of off-the-shelf multi-core computing platforms at fine-grained time-scales, especially in a growing number of complex cyber-physical applications ranging from autonomous vehicles [1] to real-time hybrid structural testing (RTHS) [2]. To conduct RTHS experiments involving hundreds-of-degrees of freedom finite element models that must be computed at millisecond time-scales, the RTHS system in [2] relies on Federated Scheduling [3] to calculate how many cores must be dedicated to each parallel real-time task to ensure schedulability.

Federated Scheduling also has been extended to support mixed-criticality tasks [4][5]. This Mixed-Criticality Federated Scheduling (MCFS) approach can reduce pessimism in resource allocation by (1) allowing higher-criticality tasks to designate a worst-case *overload* utilization as well as a common-case *nominal* utilization, and (2) allowing lower-criticality tasks to meet their deadlines as long as higher-criticality tasks only require their nominal utilizations, but still ensure that higher-criticality tasks will meet their deadlines even if they end up requiring their overload utilizations.

Whenever a higher-criticality task exceeds its nominal utilization, MCFS reallocates an appropriate number of cores to meet that additional demand and ensure it still meets its deadline. Although MCFS tries to do this gracefully by minimizing how many lower-criticality tasks are impacted, one or more lower-criticality tasks may become unschedulable.

In this paper, we show how recent advances in elastic scheduling of parallel real-time tasks [6] can be combined with the MCFS approach, to increase flexibility of lower-criticality tasks to degrade gracefully whenever a higher-criticality task exceeds its nominal utilization. We support parallel real-time task sets with arbitrary numbers of criticality levels, in which each task's elasticity may be temporal (varying its minimum inter-arrival time) or computational (varying its work).

The main contribution of this paper is a mixed-criticality model in which (1) each task runs at its nominal utilization until a virtual deadline is overrun, (2) tasks at the criticality level where the overrun occurred then run at a (higher) overload utilization, and (3) tasks with criticality levels below that at which the overrun occurred run with monotonically non-increasing (or if possible even decreasing) utilizations that are no larger than their nominal utilizations, which can adapt them elastically to remaining resources, versus being dropped.

Section II discusses related work, including background information on mixed–criticality federated scheduling and elastic scheduling. Section III describes a combined elastic multi-criticality task model and an illustrative parallel real-time task set. Section IV considers different strategies for graceful degradation based on that model. Section V discusses implications of different strategies and available cores, for the task set from Section III. Section VI concludes our discussion and describes several relevant directions for future work.

II. Background and Related Work

In this section we first present background information about Federated Scheduling and Mixed-Criticality Federated Scheduling. We then summarize prior work on elastic scheduling of sequential and parallel real-time tasks.

A. Mixed-Criticality Federated Scheduling (MCFS)

Federated Scheduling [3] supports DAG-structured implicit deadline sporadic parallel real-time tasks, without requiring task decomposition or *a priori* knowledge of their internal subtask structure – instead, each task is abstracted to a concise tuple of parameters, which capture the information needed to schedule it successfully: (C_i, L_i, D_i) where C_i is the task's work, L_i is its span (a.k.a. its *critical path*), and D_i is its implicit deadline (and also its minimum inter-arrival time or period).

MCFS [4][5] extends that model to include a criticality level Z_i and for each distinct criticality level a separate work and span. For example, in a two-criticality system where $Z_i \in \{LO, HI\}$, the tuple is $(Z_i, C_i^{LO}, C_i^{HI}, L_i^{LO}, L_i^{HI}, D_i)$. MCFS gives each higher-criticality task a *virtual deadline* (an idea used in other prior work on uniprocessor [7] and multi-processor [8] real-time scheduling of mixed-criticality task sets). Such a virtual deadline captures the time at which the system must

transition into that task's given criticality level to ensure the task meets its implicit deadline, even if it exceeds its nominal utilization $U_i^{LO} = \frac{C_i^{LO}}{D_i}$.

B. Elastic Scheduling

Buttazzo et al. developed the original model for elastic uniprocessor scheduling of sequential implicit deadline tasks [9]. Instead of a single fixed *minimum inter-arrival time* (or period), in that model each task τ_i has a range of acceptable *minimum inter-arrival times* (or *periods*) , from $T_i^{(min)}$ to $T_i^{(max)}$, and an *elastic coefficient* E_i that expresses how readily it can tolerate its performance being degraded (by increasing its *minimum inter-arrival time* within that specified range). That model was extended for variable workloads by combining control performance optimization with real-time scheduling [10] and also to accommodate resource sharing [11]. Chantem et al. showed that the iterative algorithm for updating tasks' minimum inter-arrival times in [9] is equivalent to solving a constrained optimization problem expressed in terms of the tasks' ultilizations and elastic coefficients [12]. That model also was extended to tasks with constrained deadlines [13].

Our recent work [6] has extended elastic scheduling techniques to parallel real-time task sets in which each task's work or its minimum inter-arrival time may be elastic. That model allows us to represent each *temporally elastic* task as a tuple (C_i, L_i, $U_i^{(max)}$, $U_i^{(min)}$, E_i) where C_i, L_i, and E_i are the task's (fixed) work, span, and elastic coefficient, respectively, and $U_i^{(max)}$ and $U_i^{(min)}$ are its maximum (desired) and minimum (acceptable) utilizations, respectively. Each *computationally elastic* task can be represented in a similar manner, as a tuple (T_i, L_i, $U_i^{(max)}$, $U_i^{(min)}$, E_i) where T_i is the task's (fixed) minimum inter-arrival time and the other parameters have the same interpretation as for temporally elastic tasks – that is, for temporally elastic tasks we fix C_i and allow T_i to vary, while for computationally elastic tasks we fix T_i and allow C_i to vary.

Su and Zhu [14] combined elastic scheduling and mixed-criticaltiy scheduling techniques, adjusting lowest-criticality tasks periods in uniprocessor mixed-criticality systems, under an *early-release earliest deadline first* (ER-EDF) scheduling policy, to improve schedulability. Huang et al. [15] developed an optimal variable-precision approach based on 0-1 knapsack that provides computational elasticity for graceful degradation. In comparison to those approaches, the model presented in this paper (1) supports task sets in which each task may have either computational or temporal elasticity (of workload or period, respectively), (2) supports arbitrary numbers of criticality levels, and (3) does so for parallel real-time DAG tasks whose individual resource requirements exceed a single core.

III. System Model

In this section we describe how features of the task models for multi-criticality MCFS [4][5] and our prior work on computation-or-period elasticity of parallel real-time tasks [6] can be unified into a single combined elastic multi-criticality task model that supports graceful degradation of mixed-criticality parallel real-time tasks. We then describe restrictions on that model that are necessary to guarantee schedulability under the current state of the art in elastic scheduling of parallel real-time tasks, i.e., via the techniques presented in [6]. Finally, we present an illustrative example task set under that model. The model and example are used to develop and explain new elastic compression strategies for graceful degradation, as described and discussed in Sections IV and V.

A. Towards a Generalized Elastic Task Model for MCFS

Without loss of generality, we define an implicit deadline parallel real-time task model that combines features from MCFS [4][5] and our recent work on elastic scheduling of parallel real-time tasks [6]. In this combined model, each task τ_i is expressed as a tuple (Z_i, C_i, L_i, D_i, E_i) where: Z_i is the task's criticality level; C_i, L_i, and D_i are themselves (criticality-indexed) tuples expressing the task's work, span, and implicit deadline at each criticality level in the system; and E_i is the task's elastic coefficient, with the same semantics as in [6] (a higher E_i means a task's utilization is more easily changed, though an entire core at a time with federated scheduling).

Each task's Z_i value is a non-negative number less than or equal to the maximum criticality level Z^{MAX}. The number of criticality levels in the system is $Z^{MAX}+1 = |C_i| = |L_i| = |D_i|$. In a 3-criticality task set, e.g., Z^{MAX} is 2 and $\forall_i\ Z_i \in \{0, 1, 2\}$. We use C-style array notation to index individual values in C_i, L_i, and D_i, e.g., the work of task τ_i at the lowest criticality level is $C_i[0]$ and at the highest criticality level is $C_i[Z^{MAX}]$.

As in MCFS [4][5] we define the target utilization[1] of each task τ_i at any criticality level j to be: $U_i[j] = \frac{C_i[j]}{D_i[j]}$ and define the number of cores task τ_i needs in order to meet its implicit deadlines at that criticality level to be: $m_i[j] = \left\lceil \frac{C_i[j] - L_i[j]}{D_i[j] - L_i[j]} \right\rceil$.

At any system-wide criticality level j > 0, we then compress degraded tasks' utilizations as in [6] per an updated constrained optimization based on the formulation from Chantem et al. [12], for *elastic mixed-criticality parallel real-time tasks, with respect to m total cores for the entire system*): for each task τ_i whose Z_i < j, select its utilization value U_i to **minimize** $\sum_{\forall i | Z_i < j} \frac{1}{E_i}(U_i[\alpha_i] - U_i)^2$ **such that** $\forall_i\ (U_i[Z^{MAX}] \le U_i \le U_i[\alpha_i]) \wedge m \ge \sum_{k=1}^{n} m_k[j]$, where α_i is Z_i if task τ_i has overrun its virtual deadline, or 0 if it has not, and each task's required number of cores is based on its specified deadline, work, and span at that step in the compression algorithm.

Although the generality of this representation is useful (especially for future work as described in Section VI), it is able to express task sets whose schedulability cannot be guaranteed by the techniques presented in [6], and whose accommodation is therefore deferred to future research, as we describe in Section VI. For this paper we restrict the values that some of the parameters in the task model above can be assigned, so that schedulability can be guaranteed by the techniques presented in [6], as we discuss next.

B. Restrictions on Task Set Parameters

We first restrict the compression of any degraded task τ_i to the continuous range from $U_i[\alpha_i]$ down to $U_i[Z^{MAX}]$. For consistency with the task model in [6], we set each task's span to be the same at all criticality levels: $\forall$ i,j,k $L_i[j] = L_i[k]$. We

[1] $U_i[j]$ denotes τ_i's *target* utilization at criticality level j, and simply U_i denotes the utilization chosen for τ_i by the compression algorithm.

next restrict each task to be either only temporally elastic or computationally elastic (but not both), by setting the work values of each temporally elastic task τ_v to be the same and setting the implicit deadline values of each computationally elastic task τ_w to be the same: $\forall$ x,y $C_v[x] = C_v[y]$ vs. $\forall$ x,y $D_w[x] = D_w[y]$ respectively.

In addition to these fundamental restrictions, we now discuss how other parameters of the task set are calculated and used in this combined approach. We begin by designing utilization semantics that are consistent with both mixed-criticality guarantees and graceful degradation objectives.

We assign virtual deadlines as in [4][5]. Since exceeding a virtual deadline requires adaptation of current jobs with respect to their current periods or workloads (either gaining or losing cores), we require that for computationally elastic tasks the work of a higher utilization version strictly extends the work of a lower utilization version of that same task – that is, a job's completed work counts towards all its subsequent utilizations.

At run-time, a current system-wide criticality level is set to 0 at system initialization, and is tracked and updated dynamically at run-time. Whenever a task exceeds its virtual deadline, an event noting that task's overrun is delivered to a scheduler, which as in [6] runs on its own dedicated core and is responsible for triggering adaptation by updating scheduling parameters in shared memory and then notifying tasks. The scheduler compares the criticality of the task noted in the event to the current system wide-criticality, and if the system-wide criticality is less than that of the task it is increased to that level and elastic adaptation is triggered. The scheduler resets the current system-wide criticality level to 0 at the end of each hyperperiod[2] of the task set, and the virtual deadlines of tasks whose specified criticality levels are greater 0 are re-activated within the new hyperperiod.

As in MCFS [4][5] each task τ_i operates at its nominal utilization[3] $U_i[0]$ whenever the system-wide criticality level either is below that task's specified criticality level Z_i, or (for lowest-criticality tasks) both it and the task's specified criticality level are 0: $\forall\, i,j \mid_{0=j=Z_i \vee j<Z_i} U_i[j] = U_i[0]$.

As in MCFS [4][5], except for tasks at the lowest criticality level, each task τ_i is allowed to operate at its higher overload utilization $U_i[Z_i]$ whenever the system-wide criticality level is equal to its specified criticality level Z_i :

$$\forall\, i \mid_{Z_i>0} U_i[Z_i] > U_i[0] .$$

So far, the task model adheres to the semantics of the MCFS task model [4][5]: at each criticality level, utilizations are not specified for tasks whose Z_i values are below that level, and any task whose Z_i level has been exceeded is assumed to be dropped or suspended. We now allow each task to fill in successively higher criticality levels beyond its specified criticality, with monotonically non-increasing (or if at all possible ideally monotonically decreasing) utilizations that are no larger than their nominal utilizations, which may allow them to adapt elastically to remaining resources when degraded, not dropped: $\forall\, i,j \mid_{Z^{MAX}_{\geq j>Z_i}} U_i[j] \leq U_i[0]$ and $\forall\, i,j \mid_{Z^{MAX}_{>j>Z_i}} U_i[j+1] \leq U_i[j]$.

In the resulting model, tasks operate at their nominal utilizations until they overrun their virtual deadlines, at which point the system-wide criticality level is increased to their specified criticality level (unless it already is at that level or higher). Only tasks at that criticality level that have overrun their virtual deadlines will operate at their higher (overload) utilizations, while others with that specified criticality level that have not overrun their virtual deadlines will continue to operate at their lower (nominal) utilizations. At even higher system-wide criticality levels those same tasks may operate at successively lower (degraded) utilizations according to their specified elastic coefficients and the adaptation strategy chosen as we discuss further in Section IV.A. Because only tasks that are running at their nominal utilizations (i.e., tasks whose specified criticality levels are below the system-wide criticality level) need to use virtual deadlines to trigger criticality mode changes, for simplicity of the model we simply inactivate the virtual deadline detection and handling for any task whose specified criticality level is at or below the current system wide criticality level. We discuss briefly the possibility of a more nuanced treatment of this issue as future work, in Section VI.

We impose one last restriction to remain consistent with the elastic parallel real-time scheduling semantics from [6], which only pertains to *heavy tasks* whose utilization requires more than one core. In this paper we require that at each criticality level each task behaves as a *heavy task*: $\forall\, i,j \;\; U_i[j] > 1$.

C. Illustrative Example Task Set

We illustrate our new model using a 3-criticality 5-task elastic task set, with tasks at each criticality level. Table I shows each task's criticality level and elastic coefficient, as well as the numbers of cores needed by each task τ_i at each criticality level ($m_i[0]$, $m_i[1]$, and $m_i[2]$, respectively).

TABLE I. EXAMPLE 3–CRITCALITY 5-TASK ELASTIC TASK SET

Task	E_i	Z_i	$m_i[0]$	$m_i[1]$	$m_i[2]$
τ_1	5	0	8	6	4
τ_2	4	0	6	4	2
τ_3	3	1	4	6	2
τ_4	2	1	4	6	2
τ_5	1	2	2	2	6

The light grey region in Table I shows the numbers of cores each task τ_i needs when it is operating at its nominal utilization $U_i[0]$. The next darker grey region shows the number of cores needed by each task at its overload utilization $U_i[Z_i]$. The darkest grey region shows the minimum numbers of cores each task needs at each criticality level beyond its specified criticality level Z_i. At system-wide criticality level 0 tasks τ_1 and τ_2 use 8 cores and 6 cores respectively, tasks τ_3 and τ_4 each use 4 cores, and task τ_5 uses 2 cores. At criticality level 1 tasks τ_3 and τ_4 each may need two additional cores (if both overrun their virtual deadlines) but tasks τ_1 and τ_2 only will need 6 cores and 4 cores respectively, and thus together can give up as

[2] The hyperperiod is calculated using each task's *current* D_i value.

[3] We do not allow tasks to have higher utilizations at intermediate criticalities below their specified level, and defer that to future work.

many as 4 cores. At criticality level 2 task τ_5 needs four additional cores, tasks τ_3 and τ_4 each can give up four cores (compared to their overload utilizations) and together tasks τ_1 and τ_2 can surrender up to 8 cores total compared to their maximum desired utilizations.

IV. Elastic Compression for Graceful Degradation

The elastic multi-criticality task model described in Section III allows different approaches to degrading the utilizations of tasks whose criticality levels have been exceeded, to reallocate resources to tasks whose utilizations are at their overload levels or to accommodate limited allocations of processor cores. We identify two such strategies, one in which all degraded tasks are compressed elastically together at once, and one in which compression proceeds from tasks with lower specified criticalities to tasks with higher ones.

We note that since each task is given its own separate set of cores, by design in the model presented in Section III the progress of each task is independent of the progress of the other tasks, *except to the extent that cores may be taken from one task and given to another*. Thus, the maximum desired utilization of any task τ_j that has not missed its virtual deadline is its nominal utilization $U_j[0]$, while any task τ_k that has overrun its virtual deadline is assumed to need its higher overload utilization $U_k[Z_k]$, *independent of the current system-wide criticality level*. The maximum range over which the utilization of any task τ_i can be adapted elastically is therefore from its maximum desired utilization ($U_i[Z_i]$ if it has overrun its virtual deadline, or $U_i[0]$ if it has not) down to its absolute lowest minimum acceptable utilization $U_i[Z^{MAX}]$, with such adaptation governed by its elastic coefficient E_i and the alternative elastic compression strategies we discuss next.

A. Common Structure of Both Strategies

Both strategies first select the set of tasks that should be compressed: all tasks whose specified Z_i values are lower than the current system-wide criticality level. Each strategy starts by compressing that set at the current system wide criticality level (according to that strategy's compression policy), and if that is insufficient successively (1) changes the minimum acceptable utilization of each task to be its (monotonically non-increasing) utilization value at the next higher criticality level and (2) applies its compression policy again, until utilizations are reached at which the task set fits within the allocated number of cores. Under either of these strategies, elastic compression is used only to make degradation of lower-criticality tasks as graceful as possible. If sufficient cores cannot be released elastically at a given criticality level, both strategies revert to an approach similar to that used in MCFS [4][5], by selectively dropping tasks in order (from highest to lowest utilizations at the lowest criticality level, then highest to lowest utilizations at the next higher criticality level, etc.), until the (respectively nominal, overload, or degraded) resource requirements of the remaining tasks can be met.

B. Criticality-Insensitive Compression of Degraded Tasks

The first strategy exploits the elasticity of all tasks that were selected for compression at once, in a manner that is *insensitive to their specified individual criticality levels*. This strategy takes all of the selected tasks together and adjusts their utilizations based on their elastic coefficients, as in [6]. For example, in the task set shown in Table I, at system-wide criticality level 2 tasks τ_1, τ_2, τ_3 and τ_4 would all compress their utilizations together at once to release cores that would be given to task τ_5. Although in that example the lowest criticality tasks have the highest elastic coefficients, it also is possible to have a task set in which lower criticality tasks are less elastic.

C. Criticality-Sensitive Compression of Degraded Tasks

The second strategy is *sensitive to the specified individual criticaltiy levels* of the tasks selected for compression, and takes utilization away preferentially from lower criticality tasks, and only when their elasticity is exhausted (i.e., they have reached their lowest specified utilizations) exploits the elasticity of higher-criticality tasks. For example, in the task set shown in Table I, at system-wide criticality level 2 tasks τ_1 and τ_2 would first compress their utilizations together at once (according to their respective elastic coefficients), and only if enough resources were not released when both tasks τ_1 and τ_2 reached their minimum acceptable utilizations would tasks τ_3, and τ_4 then compress their utilizations together at once (again according to their respective elastic coefficients) to release the remaining cores that would need to be given to task τ_5.

V. Discussion

A key issue with the adaptation strategies described in Section IV is how many processor cores overall are available to the task set. With the increasing applicability and relevance of real-time virtualization in particular [16][17], a task set may be run within a real-time virtual machine that is allocated a dedicated set of cores, and multiple real-time virtual machines (each with its own set of dedicated cores) may run within a single multi-core host machine. Within each virtual machine, some of its allocated cores also may be further dedicated to particular system services and mechanisms. Although such offloading avoids contention with the task set to improve performance, cores thus allocated are unavailable to the task set itself. For example, in addition to the core dedicated to the elastic scheduler (e.g., to ensure predictably timely notification and adaptation) other cores may be dedicated to I/O and other activities (e.g., to improve responsiveness).

It is thus essential to consider how a task set will behave at each criticality level with different total numbers of cores available, under the different elasticity strategies described in Section IV. In this section we first identify meaningful values for the number of cores allocated to a task set as a function of its utilization requirements, including a "sufficient" value we suggest as a design target when using our approach and a lower "limited" value we posit as the minimum reasonable allocation. We then consider system behavior with different numbers of cores, for the task set shown in Table 1, including using the elastic compression strategies if necessary.

A. How Many Cores to Allocate to a Task Set

We begin by considering an upper bound on how many cores would be useful to allocate to a task set, which is the sum of the cores needed by each task at its specified criticality level (which as noted above, is when each task's maximum utilization occurs). With that number of cores, no elastic

compression is needed. In the example task set shown in Table I, this upper bound is 8 + 6 + 6 + 6 + 6 = 32 cores.

A more salient number is the maximum of the sums of the cores needed by the tasks at each criticality level (e.g., the maximum of the column sums for any criticality level in Table 1). If provisioned, that number of cores would allow *all* tasks to operate at their specified utilizations at each criticality level (even if all tasks with specified criticalities equal to the curent system-wide criticality level have overrun their virtual deadlines), but may require adaptation according to the strategies drescribed in Section IV. In Table I, this number is 24 (at criticality levels 0 and 1 – criticality level 2 requires only 16 cores). We call this a "sufficient" number of cores and suggest it as a design target for provisioning cores to task sets under our combined multi-criticality elastic approach.

We identify numbers of cores less than the "sufficient" value as being "limited." Numbers of cores above the "sufficient" value (up to the useful limit noted previously) are considered "abundant" – e.g., 28 cores for the task set shown in Table 1. We note that having a "limited" allocation of cores means that in at least some criticality levels, some tasks cannot operate at even the utilizations given there and must degrade to a lower utilization given at a higher criticality level.

Within the "limited" values, the minimum total cores required at any criticality level (e.g., 16 for level 2 in Table 1) is important since below it at least some task must be suspended or dropped at every criticality level. However, this is not an acceptable design constraint for provisioning such elastic multi-criticality systems, as with it tasks still may need to be suspended or dropped in at least some criticality levels.

Rather, to avoid having to suspend or drop any tasks when the system-wide criticality is at any level, we must consider how many cores are needed at each system-wide criticality level x for: (1) the minimum acceptable utilizations of tasks whose specified individual criticality level is less than x or is 0; and (2) nominal or overload utilizations at level x of tasks whose specified individual criticality levels are greater than or equal to that level and are greater than 0. Thus, we must take the maximum value over all criticality levels: $m_{min} = \max_{0 \le x \le Z^{MAX}} \left(\sum_{z_i < x \vee z_i = 0} U_i[Z^{MAX}] + \sum_{z_i \ge x \wedge z_i > 0} U_i[x] \right)$. Note that in the task set shown in Table 1, for example, this value is 4 + 2 + 6 + 6 + 2 = 20, which occurs when x is 1 (with tasks τ_1 and τ_2 at their minimum acceptable utilizations of 4 and 2 respectively, tasks τ_3 and τ_4 at their overload utilizations of 6 each, and task τ_5 at its nominal utilization of 2).

B. Operation at Different Criticality Levels

We now examine how the elastic adaptation strategies described in Section IV behave at each system-wide criticality level for the example task set shown in Table 1, with different relevant numbers of allocated cores. We identify cases where no adaptation is needed, where adaptive behavior is the same under both elastic compression strategies, and where adaptive behavior differs between the two strategies.

Criticality level 0: With 24 cores allocated to the task set illustrated in Table 1, when the system-wide criticality is at level 0, each task τ_i will run at its nominal utilization $U_i[0]$ until (and unless) a higher-criticality task (whose specified Z_i value is greater than 0) exceeds its virtual deadline, at which point the system-wide criticality level is increased to that tasks's specified Z_i value. Adding more cores has no effect on the behavior since all tasks highest desired utilizations at that criticality level are already met with 24 cores. However, with only 20 cores when the system is initialized with criticality level 0, the number of cores allocated is below the 24 cores needed at that criticality level for the task set shown in Table 1. The elastic compression protocol selects tasks τ_1 and τ_2 as the set it will try to compress, and updates their minimum acceptable utilizations to be their utilizations at criticality level 1, which successfully fits the task set's utilization requirements within 20 cores. Since only tasks withh Z_i values of 0 are selected, both compression strategies give the same results.

Criticality level 1: When the system-wide criticality level is 1, with 24 cores allocated if both tasks τ_3 and τ_4 have overrun their virtual deadlines (or 22 cores if only one of them has) each task τ_i in Table 1 will run at its specified utilization at that criticality level, $U_i[1]$, until (and unless) task τ_5 (whose specified Z_i value is 2) exceeds its virtual deadline, at which point the system-wide criticality level is increased to 2. With 28 cores allocated, tasks τ_1 and τ_2 will run at their desired maximum (nominal) utilizations, which are higher than their specified (degraded) utilizations at criticality level 1.

However, with less than 24 cores, if both tasks τ_3 and τ_4 have overrun their virtual deadlines (or less than 22 cores if one of them has not) the elastic compression protocol is again invoked at system-wide criticality level 1, since a total of 24 cores (or 22 respectively if either task τ_3 or τ_4 has not overrun its virtual deadline) is needed at that criticality level. The protocol again selects tasks τ_1 and τ_2 as the set it will try to compress, and updates their minimum acceptable utilizations to be their specfied utilizations at criticality level 2, which allows the task set to fit within that number of cores, after elastic compression. Since only tasks with Z_i values of 0 are selected, both elastic compression strategies again give the same results.

Criticality level 2: With 20 or more cores, when the system-wide criticality level is 2, each task τ_i in Table 1 is able to run at or above its specified utilization at that criticality level, $U_i[2]$. Note that unless task τ_3 (respectively τ_4) has overrun its virtual deadline its maximum desired and minimum acceptable utilizations are the same and it will not factor into the compression algorithm from [6]. If neither task τ_3 nor task τ_4 has overrun its virtual deadline, only tasks τ_1 and τ_2 will be compressed and both strategies will produce the same results.

With 20 cores available, and tasks τ_3 and task τ_4 having both overrun their virtual deadlines, the two compression strategies will produce different results. Both strategies give task τ_5 6 cores, to meet its overload utilization requirements. The *criticality-insensitive compression strategy* meets task τ_1's minimum acceptable utilization with only 4 cores *since it has the highest elastic coefficient*, gives tasks τ_2 and τ_3 each 3 cores (1 more than is needed for each of their respective minimum acceptable utilizations), and gives task τ_4 4 cores (2 more than is needed for its minimum acceptable utilization). The *cricality-sensitive compression strategy* will give tasks τ_1 and τ_2 only enough cores to meet their minimum acceptable

utilizations (4 and 2 respectively), while tasks τ_3 and τ_4 are each given 4 cores (2 more than their minimum acceptable utilizations, but 2 less than their maximum desired utilizations of 6 each since both have overrun their virtual deadlines).

VI. Conclusions and Future Research Directions

In this paper we have presented a new multi-criticality elastic task model that combines features from Mixed-Cricitality Federated Scheduling [4][5] and elastic scheduling of parallel real-time tasks [6]. For now we have restricted the use of that model so that the elastic adaptation of lower-criticality tasks can be guaranteed to meet deadlines and to optimize how utilizations are adjusted elastically. We have presented and discussed alternative strategies to manage elastic compression, and have illustrated them via an example task set.

To remove the first restriction in Section III.B, we plan to allow spans (as well as work) of computationally elastic tasks to vary across criticality levels. We also will extend our model so that any task can be both computationally and temporally elastic. This would allow us to remove the restrictions in Section III.B that required work values to be the same within each temporally elastic task, and required implicit deadlines to be the same within each computationally elastic task.

We also will support adapting the system-wide criticality level downward inside each hyperperiod, to free up resources for lower-criticality tasks as soon as possible. For example, since (1) each parallel real-time task has its own cores, (2) virtual deadlines are used to track overruns, and (3) each task's next release is at its deadline, when a task overruns its virtual deadline, a counter at its criticality level is incremented, and when a task finises execution the counter at its criticality level is decremented – if the counter at the current system-wide criticaltiy level decreases to 0, the system-wide criticality level decreases to the next criticality level with a non-zero counter (or to 0 if all counters are 0). Recalculating virtual deadlines for tasks with criticality levels between the new system-wide criticality level and the (higher) previous one may be rather nuanced in some cases, and so this is defered to future work.

To reduce pessimism inherent in core-level allocation of resources, we will explore elastic mixed-criticality models in which a task may change between light and heavy utilization. We also will investigate whether other parallel real-time scheduling models besides Federated Scheduling could be used in our approach (again to make fuller use of available cores).

Finally, another direction for extending this research is to support task models with only discrete values for each task's work and span at each criticality level, and possibly also to support only discrete inter-arrival times. This will require a different approach than the techniques presented in [6] but is potentially more realistic since especially for computational elasticity, most code (other than anytime computations) tends to have discrete execution costs rather than continuous ones.

Acknowledgments

We wish to thank the anonymous reviewers of this paper for their helpful suggestions. The research described in this paper has been supported in part by NSF grant CCF-1337218 titled "XPS: FP: Real-Time Scheduling of Parallel Tasks."

References

[1] J. Kim, H. Kim, K. Lakashmanan, and R. Rajkumar. "Parallel scheduling for cyber-physical systems: Analysis and case study on a self-driving car." 4th ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS), Philadelphia, PA, April 2013.

[2] D. Ferry, G. Bunting, A. Megareh, S. Dyk, A. Prakash, K. Agrawal, C. Gill and C. Lu. "Real-Time System Support for Hybrid Structural Simulation," ACM International Conference on Embedded Software (EMSOFT), New Delhi, India, October 2014.

[3] J. Li, J.-J. Chen, K. Agrawal, C.Lu, C. Gill, and A. Saifullah, "Analysis of federated and global scheduling for parallel real-time tasks," 26th Euromicro Conference on Real-Time Systems (ECRTS), Madrid, Spain, July 2014.

[4] J. Li, D. Ferry, S. Ahuja, K. Agrawal, C. Gill and C. Lu, "Mixed-Criticality Federated Scheduling for Parallel Real-Time Tasks," IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), Vienna, Austria, April 2016.

[5] J. Li, D. Ferry, S. Ahuja, K. Agrawal, C. Gill and C. Lu, "Mixed-Criticality Federated Scheduling for Parallel Real-Time Tasks," Real-Time Systems 53(5): Special Issue on Mixed-Criticality, Multi-Core, and Micro-Kernels, pp. 760–811, May 2017.

[6] J. Orr, C. Gill, K. Agrawal, S. Baruah, C. Cianfarani, P. Ang, and C. Wong, "Elasticity of Workloads and Periods of Parallel Real-Time Tasks," 26th International Conference on Real-Time Networks and Systems (RTNS), Poitiers/Futuroscope, France, October 2018.

[7] S. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, S. Van der Ster, and L. Stougie, "The Preemptive Uniprocessor Scheduling of Mixed-Criticality Implicit-Deadline Sporadic Task Systems," 24th Euromicro Conference on Real-Time Systems (ECRTS), Pisa, Italy, July 2012.

[8] S. Baruah, B. Chattopadhyay, H. Li, and I. Shin, "Mixed-Criticality Scheduling on Multiprocessors," Real-Time Systems 50(1), pp. 142-177, January 2014.

[9] G. Buttazzo, G. Lipari, and L. Abeni, ""Elastic Task Model for Adaptive Rate Control," 19th IEEE Real-Time Systems Symposium (RTSS), Madrid, Spain, December 1998.

[10] M. Caccamo, G. Buttazzo, and L. Sha, "Elastic Feedback Control," 12th Euromicro Conference on Real-Time Systems (ECRTS), Stockholm, Sweden, June 2000.

[11] G. Buttazzo, G. Lipari, M. Caccamo, and L. Abeni, ""Elastic Scheduling for Flexible Workload Management," IEEE Transactions on Computers 51(3), pp. 289-302, March 2002.

[12] T. Chantem, X. S. Hu, and M. D. Lemmon, "Generalized Elastic Scheduling," 27th IEEE Real-Time Systems Symposium (RTSS), Rio de Janeiro, Brazil, December 2006.

[13] T. Chantem, X. S. Hu, and M. D. Lemmon, "Generalized Elastic Scheduling for Real-Time Tasks," IEEE Transactions on Computers 58(4), pp. 480-495, April 2009.

[14] H. Su and D. Zhu, "An Elastic Mixed-Criticality Task Model and its Scheduling Algorithm," Design, Automation and Test in Europe (DATE), Grenoble, France, March 2013.

[15] L. Huang, I. Hou, S. Sapatnekar, and J. Hu, "Graceful Degradation of Low-Criticality Tasks in Multiprocessor Dual-Criticality Systems," 26th International Conference on Real-Time Networks and Systems (RTNS), Poitiers/Futuroscope, France, October 2018.

[16] S. Xi, J. Wilson, C. Lu and C.D. Gill, "RT-Xen: Towards Real-time Hypervisor Scheduling in Xen," ACM International Conference on Embedded Software (EMSOFT), Taipei, Taiwan, October 2011.

[17] S. Xi, M. Xu, C. Lu, L. Phan, C. Gill, O. Sokolsky, and I. Lee, "Real-Time Multi-Core Virtual Machine Scheduling in Xen," ACM International Conference on Embedded Software (EMSOFT), New Delhi, India, October 2014.

Decoupling Criticality and Importance in Mixed-Criticality Scheduling

Konstantinos Bletsas*, Muhammad Ali Awan*, Pedro F. Souto†, Benny Akesson‡, Alan Burns§, Eduardo Tovar*
*CISTER Research Centre and ISEP/IPP, Porto, Portugal
†University of Porto, FEUP-Faculty of Engineering and CISTER Research Centre, Porto, Portugal
‡ESI (TNO), Eindhoven, the Netherlands
§Department of Computer Science, University of York, UK

***Abstract*—Research on mixed-criticality scheduling has flourished since Vestal's seminal 2007 paper, but more efforts are needed in order to make these results more suitable for industrial adoption and robust and versatile enough to influence the evolution of future certification standards in keeping up with the times. With this in mind, we introduce a more refined task model, in line with the fundamental principles of Vestal's mode-based adaptive mixed-criticality model, which allows a task's criticality and its importance to be specified independently from each other. A task's importance is the criterion that determines its presence in different system modes. Meanwhile, the task's criticality (reflected in its Safety Integrity Level (SIL) and defining the rules for its software development process), prescribes the degree of conservativeness for the task's estimated WCET during schedulability testing. We indicate how such a task model can help resolve some of the perceived weaknesses of the Vestal model, in terms of how it is interpreted, and demonstrate how the existing scheduling tests for the classic variant's of Vestal's model can be mapped to the new task model essentially without changes.**

I. Background

A. Introduction

Mixed-criticality systems are an important niche of real-time embedded systems, their defining characteristic being the fact that computing tasks of different criticalities execute on the same hardware and share system resources[1]. The criticality of a task is a measure of the severity of the consequences of a task failing (which, in the context of real-time scheduling means missing its deadline). Indeed, some tasks missing their deadline can have catastrophic consequences, whereas other tasks occasionally missing their deadlines might only have a minor effect. For this reason, the higher a task's criticality, the more conservative (and costlier, in terms of effort, time and money) the approach employed to upper-bound that task's worst-case execution time[2].

The sharing of system resources among tasks of different criticalities, unless carefully managed, can give rise to undesirable interactions that compromise safety. For this reason, and before getting to specific scheduling arrangements, the various certification authorities generally prescribe that (i) applications of lower criticality should not be able to cause the failure of tasks of higher criticality, and (ii) when tasks of different criticality share the same resources, they must all be engineered to the same strict standard of safety as the highest-criticality task thereamong. Clearly, this is inefficient. For this reason the certification authorities, in their guidelines [4], do not insist in zero interference among mixed-criticality applications, but instead expect such interference to be carefully accounted for and adequately mitigated[3]. In the context of the WCET problem, also analysing low-criticality tasks using highly conservative and pessimistic static WCET analysis techniques, as in the case of highly-critical components, would be wasteful of processing capacity. In fact, it would defeat the purpose behind the strong industrial shift to mixed-criticality scheduling, which is to efficiently utilise today's powerful multicores and reduce costs, size, weight and power.

B. Vestal's model and its evolution

This reality motivated Vestal to propose the use of different WCET estimates with different corresponding degrees of assurance, for the same task, in different scenarios, in order to ensure *a priori* the correct temporal behavior of the system at run-time [3]. To illustrate the principle, consider a fixed-priority-scheduled system and assume that each task has both a (i) "reasonable" but, not conclusively safe, WCET estimate and (ii) a highly pessimistic, but demonstrably safe, WCET estimate. Then, when testing the schedulability of a low-criticality task, one would only need to use the "reasonable" estimates, for all higher-priority tasks, irrespectively of their criticality, as inputs to the familiar Worst-Case Response Time (WCRT) analysis [5] for the task under consideration. Conversely, testing the schedulability of a high-criticality task, one would use the respective pessimistic estimates. This initial model, coupled with a fixed task priority scheduling policy, was termed Static Mixed Criticality (SMC).

Baruah and Burns extended Vestal's initial model [6] by adding *modes* and the notion of run-time *system criticality lev-*

Work partially supported by National Funds through FCT (Portuguese Foundation for Science and Technology) within the CISTER Research Unit (CEC/04234).

[1]When tasks of different criticalities exist but are completely isolated, such systems are *multiple-criticality*, as opposed to mixed-criticality, and they constitute a different class of systems. See Footnote 1 in [1] and in [2].

[2]In Steve Vestal's words: " ... the more confidence one needs in a task execution time bound (the less tolerant one is of missed deadlines), the larger and more conservative that bound tends to become in practice." [3]

[3]For example, the CAST-32A guidelines [4], clarify that "it is therefore important to identify the interference channels that could cause interference between the software applications hosted by their MCP platform, to mitigate the effects of each of those interference channels and to verify the selected means of mitigation".

el. We will henceforth refer to their model as the *Mode-based Vestal model*. In this variant model, each task has a (design) criticality level and a set of WCET estimates – one for every criticality level not exceeding its own and non-decreasing with respect to the latter. At startup, the "system criticality level" (in reality, an index of the system mode) is initialised to the lowest task criticality. If a task exceeds its WCET for the system's current criticality level, the system stops all tasks with criticality equal to that level and increments its criticality level. This constitutes a *mode change*, upon which, all tasks with criticalities lower than the current system criticality, are idled. Coupled with fixed-priority scheduling, the mode-based Vestal model is known as Adaptive Mixed Criticality (AMC), but the model itself is orthogonal to the scheduling policy. For example, it can be coupled with EDF [7], [8].

The execution time monitoring and dropping of lower-criticality tasks that exceed their WCET estimate for a given mode was a clever idea for the following reason: If a task cannot execute for more than a certain time (corresponding to that WCET estimate) without getting dropped, then that estimate (which could even be an underestimation of its true WCET), **becomes** a provably safe, unexceedable WCET estimate for the task – it will never execute for more than that. This solves the problem of unpredictable interference on higher-criticality tasks, assuming that it is acceptable to drop lower-criticality tasks, during the mode change, in the first place. This assumption in AMC may, however, not always hold, as we will discuss later, in Section I-D.

C. Terminological issues

On a related observation, and to quell some long-standing terminology-related confusion, we note what were above referred to as "WCET estimates" (or often simply "WCETs") for a given task in different modes, are in reality execution time thresholds for triggering a switch to the next-higher mode. Except for the estimates used in the top-most mode (which need to provably upper-bound the true, but unknown WCETs), the estimates (i.e., thresholds) used in other modes are *reasonably expected*, but *not required*, to upper-bound the true WCET. However, too low a value increases the probability of a mode change (undesirable, due to the degradation of functionality entailed by dropping tasks) while a value that is too high wastes processing resources. In any case, the selection of these thresholds is up to the designer. Another terminology-related source of confusion is the use of the term "system criticality" to denote what is, essentially, an indicator of the mode. Strictly speaking, "criticality" characterises applications and their tasks. This is discussed more in Section IV.

D. Other criticisms

Criticisms are sometimes voiced about the mode-based Vestal model [9], [10], [11], [12], [13] and its compatibility with the safety standards (e.g., IEC61508, ISO26262 or DO-178C) on which system certification is based. This is partially due to Vestal's use of the term "criticality" in a looser sense than the meaning it has in the standards (and the precedent that this set in the use of the term in the acadamic literature) and partially due to more legitimate concerns.

One of the more legitimate concerns, is that dropping tasks upon a mode change simply on the basis of their criticality, is not necessarily acceptable course of action in the general case. More generally, a task's criticality is not synonymous with its *importance*, which is a different attribute – and important tasks should not be discarded.

In response to such concerns, in this work **we introduce an extension of the mode-based Vestal model, whereby the importance of a task is decoupled from its criticality, and is specified separately**. The participation of a task in different modes follows from its importance, not its criticality. Meanwhile, its criticality, which determines the degree of conservativeness in its development process, also determines (in our model, just as in the standard mode-based Vestal), the degree of conservativeness in the estimation of its WCET in the different modes that it forms part of.

Note that this use of the term "importance" is different from that employed in [14] where it is simply used as a means of differentiating between tasks of the same criticality.

E. Outline of this paper

The rest of this paper is structured as follows. In Section II, we describe this new model. In Section III, we describe how the schedulability analyses developed for the standard mode-based Vestal model can be mapped to the new decoupled model, **essentially without changes**. Subsequently, in Section IV, we discuss some of the existing criticisms to the standard mode-based Vestal model, and how our new task models addresses those. Section V offers concluding remarks.

II. The Decoupled Task Model

Consider a set $\tau \stackrel{\text{def}}{=} \{\tau_1, \ldots \tau_n\}$ of n mixed-criticality sporadic tasks. Each task τ_i has a minimum inter-arrival time T_i and a relative deadline $D_i \leq T_i$. It also has a *worst-case execution time* (WCET), whose exact value is in practice unknowable, and can only be estimated. Different estimates can be obtained for the same task by the use of different techniques:

1) **Provably safe** WCET estimates, which are obtained by formal analysis and/or static path analysis, with rigorous pessimistic assumptions. They tend to be excessively pessimistic and costly to derive, in terms of time, effort and money.
2) **Potentially unsafe** WCET estimates, i.e., *probably* but not *provably* safe. These may be derived by simplistic path analysis, or via measurements and perhaps probabilistic techniques.

Although Vestal's model has been generalised to an arbitrary number of criticality levels (denoted numerically), in this work, for simplicity we assume just two criticality levels, high (H) and low (L) – which suffice in order to illustrate the principle. A task's criticality is denoted by κ_i.

Under the variant of Vestal's model introduced with AMC, there would be two modes (L and H), with all tasks executing

in the L-mode and only the H-tasks executing in the H-mode. Instead, in our model, in addition to its criticality κ_i, each task is also characterised by (what we conventionally call) its *importance* λ_i, which in the general case is indicated numerically. In this work, however, we assume that it can be high (H) or low (L), for simplicity. This attribute, input by the designer, reflects the design and application requirements and the implication is that, if needed, a low-importance task can be "dropped" (i.e., idled, at mode change) whereas a high-importance task cannot. Therefore, under our model, which tasks execute in which mode is determined on the basis of their importance, not their criticality[4]. This is a more general model than that by Baruah and Burns [6], which can be described as a special case (namely, $\lambda_i = \kappa_i,\ \forall i$).

In accordance with the above principles, there exist 4 possible classes of tasks, corresponding to the possible combinations of criticality ($\kappa_i \in \{L,\ H\}$) and importance ($\lambda_i \in \{L,\ H\}$).

Tasks of high importance (irrespective of their criticality) can also sub-categorised into (i) tasks that are present in the system already at start-up and (ii) tasks that are only introduced after the mode change (e.g., as fail-safes for one or more tasks that were dropped). Figure 1 illustrates this model, with one task for each kind. Initially (i.e., in L-mode), there exist 4 tasks in the system (τ_1, τ_2, τ_4 and τ_5, encircled in green). All of those have different combinations of criticality and importance. In L-mode, the system must be provably schedulable as long as no task executes for more than its corresponding WCET estimate for that mode (C_i^L). However, if any of those 4 tasks exceeds its C_i^L, then a mode change is triggered. The low-importance tasks τ_1 and τ_4 are then immediately dispensed with; the high importance tasks τ_2 and τ_5 persist in the new mode. Additionally, high-importance tasks τ_3 and τ_6 are added to the system, possibly to compensate for the functionality of the dropped tasks. In the H-mode, it has to be offline-provable that no task among those present (τ_2, τ_3, τ_5 and τ_6, encircled in red), can miss a deadline, assuming that these tasks execute for up to their corresponding WCET estimate for the new mode (C_i). This requirement also applies to jobs caught in the mode transition (i.e., released before the mode change, but completing after the mode change).

Note that our model imposes no constraint between the number of (low-importance) tasks dropped at mode change and the number of new (high-importance) tasks added to the system after the mode change (e.g., to compensate for their functionality). Neither is there any constraint on how the attributes of those tasks (D_i, T_i, C_i) can be related. For convenience, a task present only in the H-mode can be equivalently modelled, for schedulability analysis purposes, as a task present in both modes, with $C_i^L = 0$. Meanwhile, for low-criticality but high-importance tasks (i.e., for which a WCET estimate C_i^H for H-mode execution needs to be defined), we think that it is reasonable to use $C_i^H = C_i^L$. The reason is that rare jobs that exceed this execution time can be dropped, even in the H-mode (because the job is not critical) but the task overall cannot be dropped (because it is important) and its next job will arrive and be executed as normal. Still, there is nothing in our model that prevents the designer from specifying some other $C_i^H > C_i^L$ for a low-criticality, but high-importance, task τ_i. As for important tasks ($\lambda_i = H$) that cannot tolerate even a single dropped job, this implies that they are in fact high-criticality and need to be specified as such by the designer (i.e., $\kappa_i = H$); their WCET estimates for the H-mode would accordingly also need to be provably safe.

Ultimately, $C_i^H \geq C_i^L$, for every task that is part of the H-mode.

[4]In [11], Esper et al. discuss examples of abstract systems where a task's criticality does not reflect its importance.

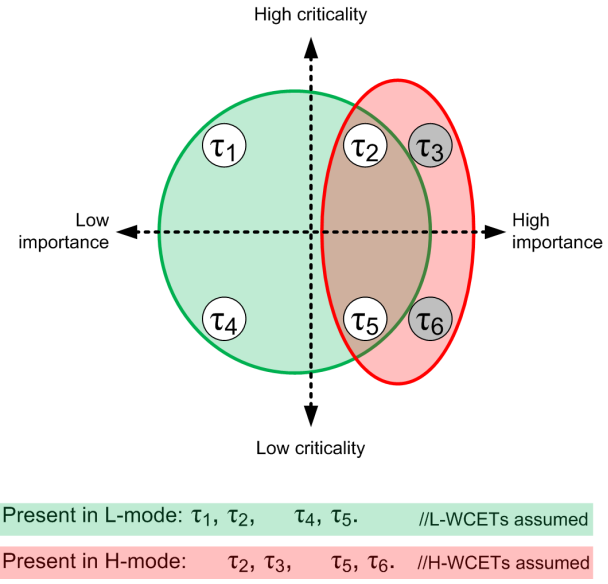


Fig. 1: A Venn diagram illustrating the different types of tasks in our task model, the modes that they can be part of and the WCET estimates used for schedulability-testing purposes.

III. Uniprocessor Analysis

Having introduced the task model, we proceed with showing how schedulability analysis formulated for the standard mode-based Vestal model can be mapped to it. The only change to the equations is that the task selector for the different modes is now the task importance. For illustration purposes, we assume a uniprocessor system and a fixed priority scheduling policy. For this case, and for the standard mode-based Vestal model, the literature offers the well-known AMC-rtb and AMC-max tests (both formulated in [15]). For schedulability testing in L-mode, for both AMC-rtb and AMC-max, a task's worst-case response time (WCRT) is upper-bounded by

$$R_i^L = C_i^L + \sum_{\tau_j \in hp(i)} \left\lceil \frac{R_i^L}{T_j} \right\rceil C_j^L \qquad (1)$$

where $hp(i)$ is the set of higher-priority tasks and C_j^L is the WCET estimate for τ_j in L-mode.

For the schedulability testing in H-mode, the WCRT equations, for AMC-rtb and AMC-max, respectively, are

$$R_i^H = C_i^H + \sum_{\substack{\tau_j \in hp(i) \\ \kappa_j = H}} \left\lceil \frac{R_i^H}{T_j} \right\rceil C_j^H + \sum_{\substack{\tau_\ell \in hp(i) \\ \kappa_\ell = L}} \left\lceil \frac{R_i^L}{T_\ell} \right\rceil C_\ell^L \quad (2)$$

and

$$R_i^H = \max(R_i^s), \ \forall s \in \{0, R_i^L\} \quad (3)$$

where

$$R_i^s = C_i^H + \sum_{\substack{\tau_\ell \in hp(i) \\ \kappa_\ell = L}} \left(\left\lfloor \frac{s}{T_\ell} \right\rfloor + 1 \right) C_\ell^L + \sum_{\substack{\tau_j \in hp(i) \\ \kappa_j = H}} \left\{ M(j,s,R_i^s) C_j^H + \left(\left\lceil \frac{t}{T_j} \right\rceil - M(j,s,R_i^s) \right) C_j^L \right\} \quad (4)$$

where

$$M(j,s,t) = \min \left\{ \left\lceil \frac{t - s - (T_j - D_j)}{T_j} \right\rceil + 1, \left\lceil \frac{t}{T_j} \right\rceil \right\} \quad (5)$$

Under the new model, what changes is that, in the degraded mode, the subset of tasks executing consists of all tasks with $\lambda_j = H$ (high importance) instead of $\kappa_j = H$ (high criticality). However, it still holds that $C_i^H \geq C_i^L$, for every task that is part of the H-mode. Correspondingly, Equation (1) need not be modified at all, whereas Equation (2) (AMC-rtb) is slightly modified to

$$R_i^H = C_i^H + \sum_{\substack{\tau_j \in hp(i) \\ \lambda_j = H}} \left\lceil \frac{R_i^H}{T_j} \right\rceil C_j^H + \sum_{\substack{\tau_\ell \in hp(i) \\ \lambda_\ell = L}} \left\lceil \frac{R_i^L}{T_\ell} \right\rceil C_\ell^L \quad (6)$$

and Equation (4) is changed to

$$R_i^s = C_i^H + \sum_{\substack{\tau_\ell \in hp(i) \\ \lambda_\ell = L}} \left(\left\lfloor \frac{s}{T_\ell} \right\rfloor + 1 \right) C_\ell^L + \sum_{\substack{\tau_j \in hp(i) \\ \lambda_j = H}} \left\{ M(j,s,R_i^s) C_j^H + \left(\left\lceil \frac{t}{T_j} \right\rceil - M(j,s,R_i^s) \right) C_j^L \right\} \quad (7)$$

with Equations (3) and (5) entirely unaffected. We typeset the modified Equations (6) and (7) with the affected terms in oversized red, in order to highlight how minimal and how straightforward the changes are. Note that the schedulability test for AMC-max is still safe even when it has to be guaranteed that jobs caught in the mode transition of tasks that are to be dropped shall not be terminated if they do not exceed their L-mode WCETs[5]. This property of AMC-max also holds under our model, if, due to design requirements, such semantics need to be enforced.

Adapting other schedulability tests for the standard mode-based Vestal model (e.g., for an EDF scheduling policy, with uniform [7] or per-task [8] deadline scaling), is analogous.

IV. Major misconceptions about the Vestal model

Having introduced our task model and shown how its schedulability analysis is available "for free" from the literature on the standard mode-based Vestal model, we are going to briefly examine to what extent its adoption settles some criticisms voiced (e.g., in [9], [10], [11], [12], [13]) at the standard mode-based Vestal model that inspired it. We also use the opportunity to highlight why some other criticisms are misframed.

A. Conflating the software assurance level of a task with the notion of importance

In systems with criticality concerns, tasks are part of one or more system components or functions, which in turn are assigned assurance levels. In the automotive domain, these are called *Safety Integrity Levels* (SILs) whereas in avionics, *Development Assurance Levels (DALs) are the equivalent concept*. Each task associated to a system component inherits the latter's SIL (DAL), with tasks belonging to multiple components (or components that are not partitioned) inheriting the highest SIL (DAL) thereof (see p. 10 in [16]). Tasks must be developed in accordance with the rules defined for their SIL/DAL.

As already mentioned, a major legitimate criticism (e.g., in [9]) on the mode-based Vestal model is that all tasks of a higher-criticality system component (i.e., higher SIL) are always given higher importance than the tasks of any lower-criticality system component (lower SIL). In other words, the importance of a task is treated as depending entirely on its criticality level. In reality though, as the critics correctly point out, some tasks of a higher-criticality component may be unimportant tasks that simply "inherited" their higher SIL due to their interaction/communication with other tasks, whose failure would be catastrophic. Conversely, there may be tasks in a lower-criticality component whose failure can be catastrophic.

As explained, our proposed model decouples the concept of criticality from that of importance. The system designer has the flexibility to specify the importance of different tasks, in accordance with the nature of the system, and decide on their placement into the different modes of operation. Hence we believe that this fully resolves the particular criticism.

[5]Quoting from the original AMC-max paper [15]: "For a possible alternative system model (in which all low criticality tasks, that have been released but not yet completed, are allowed to consume up to C(L) before being descheduled) this bound is tight."

B. Graceful degradation

Graceful degradation of the system has been recommended in safety standards (e.g., IEC61508), whereby the system is allowed to enter in a degraded mode with limited services without compromising its safety. The conventional techniques for the mode-based Vestal model aim for graceful degradation by dropping the low-criticality tasks, under the implicit assumption that these are the less important tasks. However, as explained, importance is not just a function of the criticality level, as it may also depend on the mode of operation and application context. Hence, one valid criticism is that this is a flawed attempt at graceful degradation.

Our proposed model allows the system designer to define a degradation policy based on the importance of each task, irrespective of its criticality and mode of operation. The system designer can thereby specify which tasks are to be dropped, kept or added at each mode transition.

C. WCET estimates

It has been noted (e.g., in [10], [17], [11] that nowhere do the safety standards foresee the existence of multiple WCET estimates for the same task, and that this would bring into question the compatibility of the mode-based Vestal model with these standards. To this observation, we counter that (as also noted in Section I) this is a terminological issue, even for the standard mode-based Vestal model [15]. What in the literature of the Vestal model are refered to as additional, non-provably-safe "WCET estimates", are in fact *execution time thresholds* whose excedance triggers the switch to the next mode. The consequence of setting such a threshold too high or too low (by using an execution time estimation technique that is, correspondingly, more/less conservative) is, respectively, inefficient platform utilisation vs. greater likelihood of triggering a mode change (which is also undesirable). It is a design tradeoff, and both in the standard mode-based Vestal model [15] and ours, nothing prevents the designer from specifying a provably safe WCET estimate for a task in all modes that it is part of. The fact that these estimates tend to be conventionally called (simply) "WCETs" by the people in the real-time scheduling community, obscures their true nature. However, this usage simply follows from the fact that they do behave like WCETs, when they are input into the schedulability tests that the researchers construct.

As Baruah already noted [18], even if the standards do not explicitly foresee the use of multiple execution time estimates per task, they offer no technical arguments precluding their use either, as part of a technique devised for proving the desired safety guarantees for the system. Which is why he surmises that objections to the use of multiple execution time estimates "seem in large part to be a social and cultural problem, rather than a technical one" [18].

D. Temporal and spatial isolation

The mixed criticality applications hosted on the same multicore platform can (in the absence of mechanisms preventing this) interfere with each other on multiple shared channels, including CPU, caches, memory buses, memory controllers and I/O devices [19]. Many mitigation and prevention techniques are proposed in the literature to eliminate or predictably reduce the interference among applications of different criticality. Several works already exist [20], [21], [22], [23], [24], [25], [24], [26], [27], [28], [29], [30], [31], [32], [13], [33] on ensuring temporal and spatial isolation. Some of these works already explicitly assume the mode-based Vestal model while the others can still work in its context. For instance, server-based techniques can be used to ensure temporal isolation at the CPU level [23]. The Cache Lockdown approach [34], [35] proposed in the context of the SCE framework [22] allows spatial isolation at the cache level. Similarly, the use of MemGuard [20] (and memory access regulation in general) allows for upper-bounding memory-access-related stalls and integrating them into the scehedulablity analysis.

Recently, a concern has been raised [11] that a misbehaviour in the minimum interarrival time of a task can affect the temporal isolation of a mixed-criticality system. We believe that further study is required within the context of the mode-based Vestal model to address this challenge of variation in the minimum interarrival time of a task. One way to deal with this, entirely in accordance with the spirit of the mode-based Vestal model, would be for such an inter-arrival time violation to trigger a mode change, analogously as done with execution times. Certainly, the literature contains works which consider the change of interarrival time parameters in mixed-criticality systems in different modes (e.g, [36]). Baruah also shares our view that, rather than just execution times, the theory pertaining to the Vestal model "could be used to deal with any form of inherent uncertainty and nondeterminism with regard to the run-time behavior of systems" [18].

E. Mode switch

In an industrial context, the term "criticality" refers to the level of assurance (SIL, DAL or ASIL) applied in the software development process of the safety-critical application. In the standard mode-based Vestal model, the term is "overloaded" to represent two additional concepts: the mode of operation and importance of a task. The latter comes from the (contested) assumption, in that work [15] that a task's importance is solely determined by its criticality. With respect to the former "overloaded" meaning, when a system switches from mode n to mode $n+1$, this does not really mean that the tasks change their criticality from criticality n to criticality $n+1$; it just refers to the transition in mode of operation. Despite the confusing terminology, the criticality of any application or task is not changed due to the mode transition.

Nevertheless, we recommend to other researchers in academia to use that term "mode of operation" rather than, e.g., "low system criticality" when refering to the mode, to avoid any confusion. Moreover, the decoupling of criticality from importance in our proposed model eliminates the concerns about the misuse of the term "criticality" to indicate the importance. Finally, it is worth emphasising that, in any case,

this misuse of the term has no impact on the validity of the existing analyses for the mode-based Vestal model.

V. Conclusion

We introduced a variant of the mode-based Vestal task model for mixed-criticality systems, in order to address some criticisms about the model and bridge the gap with current industrial practice. Its main feature is the decoupling of task criticality from task importance. The new model retains the essence of its predecessor (run-time robustness and efficient processor utilisation via flexible mode transition), and it remains fully backwards compatible with all its existing analyses. We also pointed out how this new model settles the concerns voiced about Vestal's mode-based model and its terminology.

To conclude, although we have no intention of discouraging anyone from exploring other techniques and paradigms, we believe that any issues or points of concern pertaining to the Vestal model, can be resolved in accordance with its spirit, and do not necessitate a break from it. We intend to work with others in the community in order to further refine the Vestal model, to not only make it more nuanced with respect to the current incarnations of the safety standards, but ultimately also to influence the drafting of future standards. Remote as this may seem today, it would not be that dissimilar to what happened with fixed-priority scheduling, which took many years and enormous effort by many people until it became endorsed by the standards and established industrial practice.

References

[1] A. Burns and R. I. Davis, "A survey of research into mixed criticality systems," Comput. Surveys, vol. 50, no. 6, pp. 82:1–82:37, Nov 2017.

[2] A. Burns and R. Davis, "Mixed criticality systems – a review (11th edition)," Department of Computer Science, University of York, UK, Tech. Rep., Aug. 2018.

[3] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in 28th RTSS, 2007.

[4] "Certification authorities software team (cast), position paper (cast-32a) multicore processors," Certification authorities in North and South America, Europe, and Asia, November 2016.

[5] M. Joseph and P. Pandya, "Finding Response Times in a Real-Time System," The Comp. J., vol. 29, no. 5, pp. 390–395, 1986.

[6] S. Baruah and A. Burns, "Implementing mixed criticality systems in Ada," in 16th Ada-Europe Conference, 2011, pp. 174–188.

[7] S. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, S. van der Ster, and L. Stougie, "The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems," in 24th ECRTS, July 2012, pp. 145–154.

[8] P. Ekberg and W. Yi, "Bounding and shaping the demand of mixed-criticality sporadic tasks," in 24th ECRTS, July 2012, pp. 135–144.

[9] A. Esper, G. Nelissen, V. Nélis, and E. Tovar, "How realistic is the mixed-criticality real-time system model?" in 23rd RTNS, ser. RTNS '15, 2015, pp. 139–148.

[10] R. Ernst and M. D. Natale, "Mixed criticality systemsa history of misconceptions?" IEEEDesign Test, vol. 33, no. 5, pp. 65–74, Oct 2016.

[11] A. Esper, G. Nelissen, V. Nélis, and E. Tovar, "An industrial view on the common academic understanding of mixed-criticality systems," J. Real–Time Syst., vol. 54, no. 3, pp. 745–795, Jul 2018.

[12] P. Graydon and I. Bate, "Safety assurance driven problem formulation for mixed-criticality scheduling," in WMC, 2013, pp. 19–24.

[13] M. Paulitsch, O. M. Duarte, H. Karray, K. Mueller, D. Muench, and J. Nowotsch, "Mixed-criticality embedded systems – a balance ensuring partitioning and performance," in 18th DSD, Aug 2015, pp. 453–461.

[14] T. Fleming and A. Burns, "Incorporating the notion of importance into mixed criticality systems," in WMC, L. Cucu-Grosjean and R. Davis, Eds., 2014, pp. 33–38.

[15] S. K. Baruah, A. Burns, and R. I. Davis, "Response-time analysis for mixed criticality systems," in 32nd RTSS, 2011, pp. 34–43.

[16] RTCA, Inc., RTCA/DO-178C. U.S. Dept. of Transportation, Federal Aviation Administration, 2012.

[17] B. Nikolić, "Mixed criticality systems – a history of misconceptions?" Invited talk. 5th Int. Workshop on Mixed Criticality Systems (WMC), Slides available at https://github.com/CPS-research-group/WMC2017/raw/master/presentations/nikolic.pptx, 2017.

[18] S. Baruah, "Mixed-criticality scheduling theory: Scope, promise, and limitations," IEEEDesign Test, vol. 35, no. 2, pp. 31–37, April 2018.

[19] D. Dasari, B. Akesson, V. Nlis, M. A. Awan, and S. M. Petters, "Identifying the sources of unpredictability in cots-based multicore systems," in 8th SIES, June 2013, pp. 39–48.

[20] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha, "Memguard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms," in 19th RTAS, April 2013, pp. 55–64.

[21] G. Yao, H. Yun, Z. P. Wu, R. Pellizzoni, M. Caccamo, and L. Sha, "Schedulability analysis for memory bandwidth regulated multicore real-time systems," Trans. Computers, vol. 65, no. 2, pp. 601–614, Feb 2016.

[22] L. Sha, M. Caccamo, R. Mancuso, J.-E. Kim, M.-K. Yoon, R. Pellizzoni, H. Yun, R. Kegley, D. Perlman, G. Arundale, and R. Bradford, "Single core equivalent virtual machines for hard realtime computing on multicore processors," University of Illinois, Tech. Rep., 2014.

[23] M. A. Awan, K. Bletsas, P. F. Souto, and E. Tovar, "Semi-partitioned mixed-criticality scheduling," in 30thARCS, 2017, pp. 205–218.

[24] M. A. Awan, K. Bletsas, P. F. Souto, B. Akesson, and E. Tovar, "Mixed-Criticality Scheduling with Dynamic Redistribution of Shared Cache," in 29th ECRTS, ser. Leibniz International Proceedings in Informatics (LIPIcs), vol. 76, 2017, pp. 18:1–18:21.

[25] M. A. Awan, P. Souto, K. Bletsas, B. Akesson, and E. Tovar, "Mixed-criticality scheduling with memory bandwidth regulation," in 55th DATE, March 2018.

[26] M. A. Awan, P. F. Souto, K. Bletsas, B. Akesson, and E. Tovar, "Worst-case stall analysis for multicore architectures with two memory controllers," in 30th ECRTS, ser. Leibniz International Proceedings in Informatics (LIPIcs), vol. 106, 2018, pp. 2:1–2:22.

[27] M. A. Awan, K. Bletsas, P. F. Souto, B. Akesson, and E. Tovar, "Mixed-criticality scheduling with dynamic memory bandwidth regulation," in 24th RTCSA, 2018.

[28] R. Mancuso, R. Pellizzoni, N. Tokcan, and M. Caccamo, "WCET Derivation under Single Core Equivalence with Explicit Memory Budget Assignment," in 29th ECRTS, ser. Leibniz International Proceedings in Informatics (LIPIcs), vol. 76. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017, pp. 3:1–3:23.

[29] K. Lampka, G. Giannopoulou, R. Pellizzoni, Z. Wu, and N. Stoimenov, "A formal approach to the WCRT analysis of multicore systems with memory contention under phase-structured task sets," J. Real–Time Syst., vol. 50, no. 5, pp. 736–773, Nov 2014.

[30] H. Kim, D. de Niz, B. Andersson, M. Klein, O. Mutlu, and R. Rajkumar, "Bounding memory interference delay in COTS-based multi-core systems," in 20th RTAS, April 2014, pp. 145–154.

[31] M. Chisholm, B. C. Ward, N. Kim, and J. H. Anderson, "Cache sharing and isolation tradeoffs in multicore mixed-criticality systems," in 36rd RTSS, Dec 2015, pp. 305–316.

[32] M. Chisholm, N. Kim, S. Tang, N. Otterness, J. H. Anderson, F. D. Smith, and D. E. Porter, "Supporting mode changes while providing hardware isolation in mixed-criticality multicore systems," in 25th RTNS, ser. RTNS '17, 2017, pp. 58–67.

[33] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha, "Memory access control in multiprocessor for real-time systems with mixed criticality," in 24th ECRTS, 2012, pp. 299–308.

[34] R. Mancuso, R. Dudko, E. Betti, M. Cesati, M. Caccamo, and R. Pellizzoni, "Real-time cache management framework for multi-core architectures," in 19th RTAS, 2013, pp. 45–54.

[35] R. Mancuso, R. Pellizzoni, M. Caccamo, L. Sha, and H. Yun, "WCET(m) estimation in multi-core systems using single core equivalence," in 27th ECRTS, July 2015, pp. 174–183.

[36] M. Jan, L. Zaourar, and M. Pitel, "Maximizing the execution rate of low-criticality tasks in mixed criticality systems," in WMC, 2013.

Journal-Never-Presented: Utilization-Based Scheduling of Flexible Mixed-Criticality Real-Time Tasks

Gang Chen, Nan Guan, Di Liu, Qingqiang He, Kai Huang, Todor Stefanov, Wang Yi

***Abstract*—In this paper, we propose a more realistic mixed-criticality model, called the flexible mixed-criticality (FMC) model, in which mode-switch of high-criticality tasks are independent. In this new model, only the overrun task itself is assumed to exhibit high-criticality behavior, while other high-criticality tasks remain in the same mode as before. The guaranteed service levels of low-criticality tasks are gracefully degraded with the overruns of high-criticality tasks. We derive a utilization-based technique to analyze the schedulability of this new mixed-criticality model under EDF-VD scheduling. During run time, the proposed test condition serves an generalized criterion for service degradation of low-criticality tasks. Finally, we show the generalization and dominance of FMC when compared to the existing EDF-VD based scheduling.**

I. Introduction

Mixed-criticality models are an emerging paradigm for the design of real-time systems because of their significantly improved resource efficiency. There is a large body of research work on specifying and scheduling mixed-criticality systems (see [2] for a comprehensive review). However, to ensure the safety of high-criticality tasks, the classic MC model applies conservative restrictions to the mode-switching scheme. In the classic MC model, whenever *any* high-criticality task overruns, *all* low-criticality tasks are immediately abandoned and *all other* high-criticality tasks are assumed to exhibit high-criticality behaviors. Although there has been some research on solving the first problem, i.e., *statically* reserving a certain degraded level of service for low-criticality execution, to our knowledge, little work has been done to date to address the second problem except for the recent published work [3].

In the work [1], we propose a flexible MC model (denoted by **FMC** for short) on a uni-processor platform, in which the two aforementioned issues are addressed in a combined manner. In FMC, the mode switches of all high-criticality tasks are independent. A single high-criticality task that violates its low-criticality WCET triggers only itself into high-criticality mode, rather than triggering *all* high-criticality tasks. In this manner, FMC only requires to book part resources for overruned high-criticality task, rathern than for all high-criticality tasks. More importantly, low-criticality tasks can be adaptively penalized according to run-time overrun workload, by which the system workload can be balanced with minimal service degradation for low-criticality tasks.

As the main contribution of the work [1] , we study the schedulability of the proposed FMC model under EDF-VD scheduling. A utilization-based schedulability test condition is derived by integrating the independent triggering scheme and the adaptive service level tuning scheme. A formal proof of the correctness of this new schedulability test condition is presented. In addition, we explore the feasible region of the virtual deadline factor for this new MC model. Finally, we show the generalization and dominance of FMC when compared to the existing EDF-VD based scheduling.

The work has been published in [1]

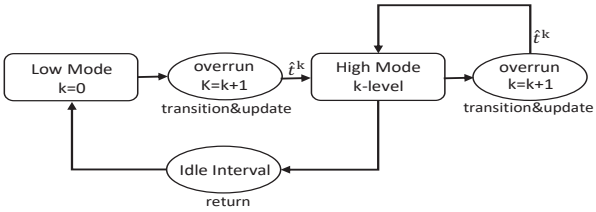


Figure 1. Execution semantics of the FMC model.

II. FMC Model and Schedulability Anaslysis

A. Flexible Mixed-Criticality Model

Now, we provide the definition of FMC model, which extended from classic EDF-VD MC scheduling. The execution semantics of FMC can be depicted as Fig. 1.

Execution Semantics:All tasks in γ start in 0-level high-criticality mode (i.e., low-criticality mode). As long as no high-criticality task violates its C_i^{LO}, the system remains in 0-level high-criticality mode. When one job of a high-criticality task *that is being executed in low-criticality mode* overruns its C_i^{LO}, this high-criticality task immediately switches into high-criticality mode. All other high-criticality tasks still remain in the **same** mode as before. At the k^{th} transition point (corresponding to time instant $\hat{t}^k$ in Fig. 1), the execution budget of every high-criticality task are updated as $z_i^k \cdot c_i^{LO}$ to compensate the overrun of high-criticality tasks ($z_i^k \leq z_i^{k-1} \leq 1$). When the system detects an idle interval [4], [5], the system will transition back into low-criticality mode.

B. Schedulability Analysis Results

The schedulability analysis of FMC requires to guarantee the schedulability on low-criticality mode and high-criticality mode, which are summerized in Thm. 1 and Thm. 2. In low-criticality mode, the system behaviors in FMC are exactly the same as in EDF-VD [6].

Theorem 1: The following condition is sufficient to ensure that EDF-VD can successfully schedule all tasks in low-criticality mode:

$$u_{LO}^{LO} + \frac{u_{HI}^{LO}}{x} \leq 1 \tag{1}$$

In Thm. 2, we provide the answer to the question of how much execution budget can be reserved for low-criticality tasks while ensuring a schedulable system for mode transitions. Without loss of generality, we consider a general transition case in which the system transitions from $(k-1)$-level high-criticality mode to k-level high-criticality mode.

Theorem 2: The system is in $(k-1)$-level high-criticality mode. For the k^{th} mode-switching point $\hat{t}^k$, when high-criticality task $\tau_{\hat{t}^k}$ overruns, the system is schedulable at $\hat{t}^k$ if the following conditions are satisfied:

$$u_{LO}^{k} \leq u_{LO}^{k-1} + \frac{\frac{u_{\hat{t}^k}^{LO}}{u_{HI}^{LO}}(1 - u_{LO}^{LO}) - u_{\hat{t}^k}^{HI}}{(1-x)} \tag{2}$$

$$z_i^k \leq z_i^{k-1} \quad (\forall \tau_i \in \gamma_{LO}) \tag{3}$$

where $u_{\hat{t}^k}^{LO}$ and $u_{\hat{t}^k}^{HI}$ denote low and high utilization, respectively, for the high-criticality task $\tau_{\hat{t}^k}$ that undergoes a mode switch at $\hat{t}^k$.

Thm. 2 only provides scheduability test condition only for a single transition. However, the feasibility of algorithms that off-line determines whether a task set is schedulable by FMC-MCL under arbitrary sequences of mode switches is not known yet. Thm. 3 provides a system level scheduability test condition for a task set.

Theorem 3: Given the mandatory utilization u_{LO}^{man}, any x that satisfies the following condition can guarantee that a feasible solution as determined by Thm. 2 can always be found during run time.

$$(1-x)(u_{LO}^{LO} - u_{LO}^{man}) + \sum_{\tau_i \in \gamma_{HI}^*} \phi(\tau_i) \geq 0 \tag{4}$$

where $\phi(\tau_i) = \frac{u_i^{LO}}{u_{HI}^{LO}}(1 - u_{LO}^{LO}) - u_i^{HI}$ and $\gamma_{HI}^* = \{\tau_i \in \gamma_{HI} | \phi(\tau_i) \leq 0\}$.

III. New Results

In [1], the experiment results show FMC achieves almost the same schedulability performance, but still be inferior to EDF-VD. Now, we will show this inferiority can be fixed by following new feasibility condition. At first, we show the old feasibility condition in [1] is the same as the condition of EDF-AD in [3]. This equivalence is shown in Thm. 4. Then, based on similar observation in [3], we can also derived a new feasibility condition which domains the regular EDF-VD, as shown in Propo. 1.

Theorem 4: Given a task set with $u_{LO}^{man} = 0$, then FMC is schedulable if:

$$u_{LO}^{LO} + \frac{u_{HI}^{LO}}{x} \leq 1 \tag{5}$$

$$x \cdot u_{LO}^{LO} + \sum_{\tau_i \in \gamma_{HI}} \max(\frac{u_i^{LO}}{x}, u_i^{HI}) \leq 1 \tag{6}$$

Proof. According to Thm. 3 in [1], the old feasibility condition with $u_{LO}^{man} = 0$ can be represented as:

$$(1-x)u_{LO}^{LO} + \sum_{\tau_i \in \gamma_{HI}^*} \phi(\tau_i) \geq 0 \tag{7}$$

According to off-line step in FMC-EDF-VD, we determine x as $\frac{u_{HI}^{LO}}{1-u_{LO}^{LO}}$. Therefore, we have:

$$\begin{aligned}
&(1-x)u_{LO}^{LO} + \sum_{\tau_i \in \gamma_{HI}} \min(\frac{u_i^{LO}}{x} - u_i^{HI}, 0) \geq 0 \qquad (8)\\
\Leftrightarrow &(1-x)u_{LO}^{LO} + \sum_{\tau_i \in \gamma_{HI}} (\min(\frac{u_i^{LO}}{x}, u_i^{HI}) - u_i^{HI}) \geq 0\\
\Leftrightarrow &x \cdot u_{LO}^{LO} + u_{HI}^{HI} \leq u_{LO}^{LO} + \sum_{\tau_i \in \gamma_{HI}} \min(\frac{u_i^{LO}}{x}, u_i^{HI})\\
\Leftrightarrow &x \cdot u_{LO}^{LO} + u_{HI}^{HI} + \frac{u_{HI}^{LO}}{x} - \sum_{\tau_i \in \gamma_{HI}} \min(\frac{u_i^{LO}}{x}, u_i^{HI}) \leq u_{LO}^{LO} + \frac{u_{HI}^{LO}}{x} = 1 \qquad (9)\\
\Leftrightarrow &x \cdot u_{LO}^{LO} + \sum_{\tau_i \in \gamma_{HI}} \left(u_i^{HI} + \frac{u_i^{LO}}{x} - \min(\frac{u_i^{LO}}{x}, u_i^{HI})\right) \leq 1
\end{aligned}$$

According to $\max(a,b) = a+b-min(a,b)$, we have Eqn. (6).

□

Above, we have demonstrate FMC is equivalent to EDF-AD. Now, we will present a new feasibility condition to show FMC is equivalent to EDF-AD-E in [3].

Theorem 5: Given a task set with $u_{LO}^{man} = 0$, then FMC is schedulable if:

$$u_{LO}^{LO} + \sum_{\tau_i \in \gamma_{HI}} \min(\frac{u_i^{LO}}{x}, u_i^{HI}) \leq 1 \tag{10}$$

$$x \cdot u_{LO}^{LO} + u_{HI}^{HI} \leq 1 \tag{11}$$

Proof. According to Lemma. 6.3, Eqn. (10) ensures the low-criticality schedulability. By Eqn. (8), we have:

$$x \cdot u_{LO}^{LO} + u_{HI}^{HI} \leq u_{LO}^{LO} + \sum_{\tau_i \in \gamma_{HI}} \min(\frac{u_i^{LO}}{x}, u_i^{HI}) \leq 1 \tag{12}$$

□

According to [3], EDF-AD-E dominates EDF-VD in terms of MC-schedulability. FMC is equivalence to EDF-AD-E. Therefore, FMC dominates EDF-VD.

Property 1: FMC dominates EDF-VD in terms of MC-schedulability.

Generalization: Compared to [3] which only targets on task dropping-off strategies, the work [1] provides an general criterion for run-time service level tuning. By checking the conditions in Thm. 2, one can determine how much utilization can be reserved for low-criticality task execution to compensate for the overruns. In general, various tuning strategies can be specified by the user as long as the condition in Thm. 2 is satisfied during run time. Task dropping-off strategy used in [3] can be considered a special case by assigning $z_i^k = 0$ for dropped tasks.

References

[1] C. Gang *et al.*, "Utilization-based scheduling of flexible mixed-criticality real-time tasks," *IEEE Transaction on Computers*, 2017.

[2] A. Burns *et al.*, "Mixed criticality systems-a review," *University of York, Technical Report*, 2015.

[3] J. Lee *et al.*, "Mc-adapt: Adaptive task dropping in mixed-criticality scheduling," *ACM Transactions on Embedded Computing Systems*, 2017.

[4] A. Burns *et al.*, "Towards a more practical model for mixed criticality systems," in *1st International Workshop on Mixed Criticality Systems*, 2013, pp. 1–6.

[5] F. Santy, L. George, P. Thierry, and J. Goossens, "Relaxing mixed-criticality scheduling strictness for task sets scheduled with fp," in *2012 24th Euromicro Conference on Real-Time Systems*, 2012.

[6] S. Baruah *et al.*, "The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems," in *2012 24th Euromicro Conference on Real-Time Systems*, 2012.

ERDN 2018

Proceedings of the 1st Workshop on Efficient Real-Time Data Networks

Edited by
Huayu ZHANG and Hui LI

Organisers

Program Chairs

Huayu Zhang and Hui Li

Steering Committee

Liaini (Chair), Huawei Network Technology Laboratory, China
Zhe Lou, Huawei German Research Center, Germany
Yan SUN, Queen Mary University of London, UK
Xiaodong Xu, Beijing University of Posts and Telecommunications, China
Kaixuan Xing, Peking University, China

Program Committee

Zhigang Zhu, Huawei Network Technology Laboratory, China
Tong Yang, Peking University, China
Wenfei Wu, Tsinghua University, China
Xiaodong Xu, Beijing University of Posts and Telecommunications, China
Gerard Parr, University of East Anglia, UK
Guanrong Chen, City University of Hong Kong, Hong Kong
Yan SUN, Queen Mary University of London, UK
Dongsu Han, School of Electrical Engineering, KAIST, Korea
Jonathan Loo, University of West London, UK
Changchuan Yin, Beijing University of Posts and Telecommunications, China
Chris Phillips, Queen Mary University of London, UK
Nampuraja Enose, Advanced Engineering Group at Infosys Limited, Germany
Juergen Jasperneite, Fraunhofer IOSB-INA, Germany
Lukasz Wisniewski, OWL University of Applied Sciences, Germany
Vladimir Braverman, Johns Hopkins University, USA
Liqiang Wang, University of Central Florida, USA
Balajee Vamanan, University of Illinois at Chicago, USA
Wenjun Li, Peking University, China

ERDN 2018 PC Co-chairs' Message

Welcome to the 1st Workshop on Efficient Real-time Data Networks (ERDN 2018). ERDN 2018 is to be held in conjunction with The 39th IEEE Real-Time Systems Symposium (RTSS 2018) in Nashville, Tennessee, USA between 11th-14th December 2018. On behalf of the program committee, we welcome you to Nashville.

The booming development of Internet in the past decades has changed the world substantially. The future trend of the next decade is to offer deeper inter-connection to traditional industry. The vision of Industry 4.0 makes us realize that IoT, Internet of Vehicles, AR/VR, machine-to-machine, AI and so on have put more stringent requirement on real-time information exchange. Unlike tradition applications like Voice over IP, Video over IP, Online Game which require end-to-end millisecond-level latency, the requirement of these emerging applications is usually at microsecond or even nanosecond level. Therefore, efficient real-time network without geographic limitation becomes the key to success. This workshop aims to bring experts in both academia and industry to discuss the key techniques to build this kind of network. We welcome ideas on improvements based on existing methods and potential technology under research as well.

We have received quite a few interesting papers. All the papers are strictly reviewed by at least 3 reviewers and finally 4 papers are selected to make this workshop work. For example, Ahlem Mifdaoui et al. analyze the performance of Time Sensitive Networks in Heterogeneous Avionics Applications. Chonghui Ning et al. focus on the traditional packet classification by using TCAM. Meng Wan et al. use machine learning to improve real-time E-commerce serving system. Chung-Fan Yang et al. try to introduce the cache technologies to solve the latency problem in virtual machine environment. We have confidences that these papers will inspire further research in its applications in many areas.

We would like to express our deepest appreciation to all the authors for sharing their ideas and results with us, and all the members of the Program Committee and reviewers for their help in evaluating and identifying high quality papers. We are proud to thank Prof. Jean-Luc Scharbarg for his keynote on Trends in avionics communication systems and Leandro Soares Indrusiak for his invited talk on Evolutionary Optimisation of Real-Time NoC-based Many-Cores.
We attribute the success of ERDN 2018 to all RTSS 2018 Program, Workshop and Organization Chairs, especially Liliana Cucu-Grosjean, Robert I. Davis and Isabelle Puaut for their help and encouragement.

We thank you for your active participation and hope that you will equally enjoy the workshop and your stay in Nashville.

Huayu Zhang, Aini Li, Hui Li

ERDN 2018 Technical Program

ERDN 2018 Keynote

Jean-Luc Scharbarg

Trends in Avionics Communication Systems

BIO : Prof. **Jean-Luc Scharbarg** received the PhD in computer science from the Univer- sity of Rennes, France, in 1990 and the qualification to advice PhD students from the Universite de Toulouse, France, in 2010. He has been full professor at the Universite de Toulouse (INPT/ENSEEIHT and IRIT laboratory) since 2012 and head of the RMESS team of IRIT since 2011.

He has co-authored more than 100 papers in distinguished international conferences and journals. He has served in the program committee of numerous conferences and workshops. He has been involved in the organisation of several international workshops and conferences and he was the chair of some of them.

His current research interest concerns the design, analysis and performance evaluation of embedded networks, mainly in the context of avionics. He has worked on many projects with the aerospace industry. In particular, he has been involved in the definition and analysis of avionics networks on-board of Airbus aircrafts.

ERDN 2018 Invited Talk

Leandro Soares Indrusiak

Evolutionary Optimisation of Real-Time NoC-based Many Cores

BIO: **Leandro Soares Indrusiak** is a faculty member of University of York's Computer Science department, and a member of the Real-Time Systems (RTS) research group. His current research topics include real-time systems and networks, on-chip multi and many-cores, distributed embedded systems, high-performance computing, and several types of resource allocation problems in computing, manufacturing and transportation. He has published more than 150 peer-reviewed papers in the main international conferences and journals covering those topics (nine of them received best paper awards, the last one at DATE 2018). He has graduated nine doctoral students over the past 10 years, and currently supervises three doctoral students and three post-doc research associates.

He is or has been a principal investigator in projects funded by EU, EPSRC, DFG, British Council and industry. He serves as his department's Internationalisation Advisor, and has held visiting faculty positions in five different countries. He is a member of the EPSRC College, a member of the HiPEAC European Network of Excellence, a senior member of the IEEE, a member of the editorial board of ACM Transactions on Cyber-Physical Systems, and a member of York's Sciences Faculty Board.

He graduated in Electrical Engineering from the Federal University of Santa Maria (UFSM) in 1995, obtained a MSc in Computer Science from the Federal University of Rio Grande do Sul (UFRGS) in 1998, and was issued a binational doctoral degree by UFRGS and Technische Universität Darmstadt in 2003. Prior to his appointment at York, he held a tenured assistant professorship at the Informatics department of the Catholic University of Rio Grande do Sul (PUCRS) (1998-2000) and worked as a researcher at the Microelectronics Institute of TU Darmstadt (2001-2008).

ABSTRACT: The design space of many-core processors based on Networks-on-Chip (NoCs) is very large, posing challenges to the optimisation of such platforms when it comes to support applications with real-time guarantees. Recent research has shown that a number of inter-related optimisation problems has critical influence over the schedulability of a system, i.e. whether all its application components can execute and communicate by their respective deadlines. Examples of such optimization problems include task allocation and scheduling, communication routing and arbitration, memory allocation, voltage and frequency scaling. Given the size and complexity of such design spaces, it is unlikely that a solution can be found which is simultaneously optimal for all problems. Actually, for reasonably complex many-cores, it is already infeasible to find optimal solutions to a single one of those optimization problems, let alone to all of them. In this talk, I will

Comparative Analysis of Scheduling Strategies for Heterogeneous Avionics Applications

A. MIFDAOUI, A. FINZI, F. FRANCES and E. LOCHIN
ISAE-Supaéro / Université de Toulouse, France

***Abstract*—A homogeneous avionic communication architecture to interconnect different avionics domains may bring significant advantages, such as easier installation and maintenance in addition to reduced weight and costs. This homogeneous communication architecture needs to support heterogeneous applications, where safety-critical and best effort traffic co-exist. In this paper, we assess the pros and cons of the most relevant scheduling strategies supporting heterogeneous applications versus the main avionics requirements. Furthermore, we conduct a quantitative comparative analysis of the most promising solutions guaranteeing the main avionics requirements through a representative avionics case study. Results show that a recent shaper in Time Sensitive Networks is a promising solution in terms of performance and complexity.**

***Index Terms*—TSN, BLS, AFDX, DRR, NP-SP, avionics, QoS, Schedulers.**

I. Introduction

Avionics is a field that moved from point-to-point transmissions to high speed networks. However, this field slowly evolves due to the stringent safety requirements and the aircraft long life expectancy, around 25 to 30 years. The comparison of this lifespan against other networking fields is an interesting one. For instance, the last 30 years have seen the development of main stream Internet, from low rate 64Kbit/s to high speed Gigabit fiber connections. Concerning mobile networks, a new generation appears approximatively every 9 years. Hence, between the day in 1990 when an airliner entered into service to its retirement in 2015, a consumer download link was multiplied by 15,000 and 3 mobile network generations were developed. This highlights the stark difference between the closed avionics world, and the Internet and mobile open world.

However, linkages exist between these communities: the newest avionics network, the Avionics Full-DupleX Ethernet (AFDX) [1] is based on a technology developed for the Internet, the Switched Ethernet. The low cost and maturity, after decades of use in industrial markets, are the main advantages of this technology. There are still many technologies from the open world that could be used for avionics networks. In particular in the open world, there is a large number of scheduling strategies to multiplex heterogeneous flows within a network. In this paper, we analyse the most relevant ones to assess their potential use to define an avionics network to support heterogeneous avionics applications.

With the maturity and reliability progress of the AFDX after a decade of successful use, a homogeneous avionic communication architecture based on such a technology to interconnect different avionics domains may bring significant advantages, such as easier installation and maintenance in addition to reduced weight and costs. This homogeneous communication architecture, based on the AFDX technology, needs to support heterogeneous applications, where safety-critical and best effort traffic co-exist. Hence, in addition to the current AFDX traffic profile, called Rate Constrained (RC) traffic, at least two extra profiles have to be handled. The first, denoted by Safety-Critical Traffic (SCT), is specified to support flows with hard real-time constraints and the highest criticality, e.g., flight control data; whereas the second is for Best-Effort (BE) flows with no delivery constraint and the lowest criticality, e.g., In-Flight Entertainment traffic.

Hence, we start by presenting the avionics context through the evolution of avionics network and the main avionics requirements in Section II. Afterwards, we assess the pros and cons of the most relevant scheduling strategies supporting heterogeneous applications versus the main avionics requirements in Section III. Finally, we conduct a quantitative comparative analysis of the most promising solutions guaranteeing the main avionics requirements through a representative avionics case study in Section IV.

II. Avionics Context

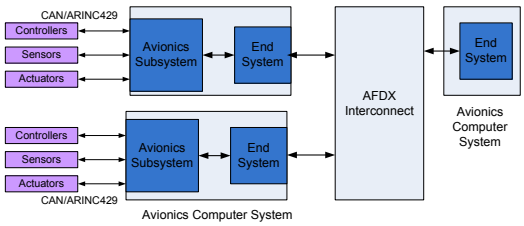


Fig. 1: Current Avionics Network

As shown in Figure 1, the current avionics network, responsible for flight control, cockpit, engines and fuel & landing gears, consists of a high-rate backbone network, the AFDX [1], to interconnect critical subsystems. Moreover, some low-rate data buses, e.g., CAN [12] or ARINC 429 [7], are still used to handle some specific avionics domains, such as the I/O process and the Flight Control Management.

The AFDX [1] network is based on Full Duplex Switched Ethernet protocol at 100Mbit/s, successfully integrated into new generation civil aircraft like the Airbus A380. This technology succeeds to support the important amount of exchanged data due to policing mechanisms added in switches and the

Virtual Link (VL) concept. The latter gives a way to reserve a guaranteed bandwidth to each traffic flow. The VL represents a multicast communication which originates at a single End-System and delivers packets to a fixed set of End-Systems. Each VL is characterized by: (i) BAG (Bandwidth Allocation Gap), ranging in powers of 2 from 1 to 128 milliseconds, which represents the minimal inter-arrival time between two consecutive frames; (ii) MFS (Maximal frame size), ranging from 64 to 1518 bytes, which represents the size of the largest frame that can be sent during each BAG. The current AFDX enables the use of both scheduling strategies: First Come First Served (FCFS) and Strict Priority (SP).

The CAN bus [12] is a 1 Mbit/s data bus that operates according to an event-triggered paradigm where messages are transmitted using a priority-based access mechanism. CAN bus works using a producer/consumer communication scheme based on unique identifier per message type. The CAN messages are broadcasted on the bus, then each CAN equipment will filter the consumed data based on the CAN identifier. The collisions on the bus are resolved following a CSMA/CR protocol (Carrier Sense Multiple Access/ Collision Resolution) thanks to the bit arbitration method.

The ARINC429 [7] is a 100 Kbit/s data bus with a point-to-point protocol. It is a mono transmitter multi receivers data bus with unidirectional communication which provides high reliability at the cost of wire weight and limited data rates.

Although this architecture reduces the time to market, it conjointly leads to inherent heterogeneity and new challenges to guarantee the real-time requirements. To enable a homogeneous architecture based on AFDX technology, we identify herein the main avionics requirements and challenges to compare the different scheduling strategies and select the most appropriate one to support heterogeneous flows on the AFDX.

The two main considered avionics requirements are as follows:

- **Predictability**: the impact of a system on an other is known and bounded. The communication architecture must behave in a predictable way, where the extended AFDX has to guarantee bounded latencies respecting the temporal constraints of the heterogeneous traffic.
- **Modularity**: this requirement is related to the flexibility and exchangeability of software and hardware components. An important step towards enhancing the avionics system modularity has been fulfilled with the adoption of the IMA approach [17], i.e., common elementary components can be configured to fit different avionic applications. This feature aims to minimise the (re) configuration and readjustment effort to facilitate system maintenance and its progress over the years. For instance, the event-triggered paradigm of the AFDX is favoring such a requirement.

Moreover, we need to deal with the main challenge of enforcing the Quality of Service (QoS) features, while limiting the impact of the highest priority traffic on the current AFDX traffic and the implementation complexity. These challenges will be denoted by **Fairness**, and **Complexity** along this paper.

III. Qualitative Analysis of Different Scheduling Strategies

Various solutions have been proposed in the literature to support heterogeneous applications in embedded systems and particularly in avionics. The first proposed solution is the simplest one, based on Strict Priority like the one specified in the AFDX. Overtime, new solutions with increased complexity were proposed, such as the ones defined in Audio Video Bridging [11] and Time-Sensitive Networking [16].

To quantify the (re)configuration effort needed by an alternative avionics communication architecture in comparison to the current AFDX standard, the considered communication paradigm is of utmost importance since the modularity level of a solution highly depends on such a paradigm. The event-triggered paradigm is known as highly flexible and facilitates the system reconfiguration, but it infers at the same time an indeterminism level and needs further proofs to verify the predictability requirement. On the other hand, the time-triggered paradigm is highly predictable, but presents some limitations in terms of system reconfigurability.

In this section, we will detail the different scheduling strategies and assess their potential ability vs the avionics requirements.The different solutions can be categorized according to the required communication paradigm, i.e., mainly time-triggered or event-triggered.

Non-Preemptive Strict Priority Scheduler The Non-Preemptive Strict Priority (NP-SP) scheduling strategy is the simplest QoS implementation with very limited complexity. Each queue has a defined priority and the scheduler dequeues the first frame of the eligible queue (a queue with enqueued traffic) with the highest priority. This scheduler is defined in the AFDX standard [1]; and due to the leaky bucket shapers in the end-systems and policers in the switches, NP-SP guarantees the predictability requirement.

NP-SP is compliant with an event-triggered paradigm, which allows a high modularity level, but it is a well-known as an unfair scheduler [18].

GPS-like Schedulers–Deficit Round Robin

The Generalized Processor Sharing (GPS) is an idealized scheduling algorithm that achieves perfect fairness: the bandwidth is shared depending on fixed weights. Many algorithms have been developed to come as close as possible to the GPS, such as the Weighted Fair Queuing (WFQ) [4] or Weighted Round Robin (WRR) [19] and Deficit Round Robin (DRR) [9]. Ordinary round-robin servicing of queues can be done in constant time. With WRR, the usual implementation consists in setting a number of frames that can be consecutively sent for each queue. The major problem, however, is the unfairness caused by possibly different packet sizes used by different flows. This flaw can be removed by using a counter to keep track of traffic transmitted as with the Deficit Round Robin (DRR). Nonetheless, these schedulers necessitate a virtual clock, which increases their implementation complexity. In [9], an AFDX network implementing the DRR has been specified and studied.

Results have shown the good performances of the proposal in terms of predictability and fairness, while increasing the implementation complexity. Moreover, like NP-SP, DRR offers a high modularity level due to its compliance with an event-triggered paradigm.

Audio-Video Bridging–Credit Based Shaper

In recent years, there has been a strong interest in the IEEE 802.1 Audio/Video Bridging (AVB) protocol, which provides end-to-end delay guarantees in Ethernet networks. AVB specifies a credit-based shaping (CBS) algorithm for real-time (RT) traffic classes A and B. Each shaped class has a credit-counter, which is replenished at a constant rate (the so-called idle slope) and consumed at the rate allowed by the port (the send slope) when data on the specific class is transferred. When the queue is empty, the credit immediately returns to 0. The different classes are scheduled using a strict priority scheduler, with the CBS preventing the starvation of lower priorities and giving bandwidth guarantees, which are good properties for mixed-criticality applications.

Concerning the predictability of CBS, the different classes are isolated from each other thanks to the counter and their associated blocking effect. However, it has been shown in [2] that the impact of the blocking effect of the AVB on the latency is high, which induces a medium predictability level for this shaper. However, the worst-case latency of unshaped lower priorities is improved due to the shaping of classes A and B, which fulfills the fairness challenge. The main drawback of the CBS is that frames cannot be transmitted if the credit is below 0, no matter the state of the other queues. This fact can cause unnecessary delays if other queues are empty. This issue has been fixed by the TSN [16] task group through different shapers.

Time Sensitive Networking–Time Aware Shaper

TAS[15] uses time-driven scheduling to manage link access between traffic classes, which makes it a good candidate for heterogeneous traffic flows. For each traffic class, the frames are transmitted according to a gate schedule at each output port: it allows frames to pass when opened, and it blocks frames when closed. The different gate schedules are programmed offline, and multiple gates can be opened at the same time. Then, the selected frames are arbitrated according to their priority levels. To prevent frames transmission when the gate is closed, TAS defines guard bands. From the start of a guard band until the gate is opened, no new frames of the corresponding class are allowed to start transmission.

Due to the gate schedule, TAS guarantees a high predictability level, but the modifications are propagated to all flows. This fact limits the TAS modularity, while inferring high implementation complexity. Additionally, when lower classes gates are opened, they are scheduled using a strict priority, which implies a low fairness.

Time Sensitive Networking–Peristaltic Shaper

The Peristaltic Shaper (PS) [14] uses a global time divided in odd and even phases to manage different traffic classes. If a shaped frame arrives in an odd (resp. even) phase, it can not be sent before the start of the next even (resp. odd) phase. The idle time can be used by other priorities. The Peristaltic Shaper has been proposed by the same task group as TAS. Hence, they have often been studied together and similar work has been done.

Similarly TAS, the use of a global time in PS implies a high predictability level but a negative impact on its modularity and implementation complexity: a flow modification can impact the calculation of odd and even phases not only along its path, but also on other flows paths. However, due to the initial waiting time caused by the odd and even phases, lower priority flows may be sent more quickly than under Static Priority scheduler, which makes Peristaltic Shaper an interesting solution in terms of fairness.

Time Sensitive Networking–Urgency-based Scheduler

The main idea of the Urgency-based Scheduler (UBS) [13] is a separation between per flow and per queue. The conceptual separation of per flow queue and state provides per flow shaping at every hop for flow aggregated in the queues. This concept is called $interleaved\ shaping$. This significantly reduces the algorithmic complexity by limiting the number of required queues. Hence the first step when a new frame arrives in the output port is to select the appropriate queue depending on the priority of the flow and its ”urgency” as decided by an interleaving algorithm.

This scheduler is still new, so little research has been done yet. In [13], the scheduler is presented, simulations and timing analysis are performed. The results show high link utilisation and low delays. They also conclude that the implementation complexity is low, in part because they assume the queue selection process is already implemented in the switches thanks to the standardisation of 802.1Qci-Per-Stream Filtering and Policing. But, while implementing it in higher layer is simple, implementing at the hardware level is much more complex.

Time Sensitive Networking–Burst Limiting Shaper

Presented in [8], the BLS is a credit-based shaper that has been characterized in [8] by an upper threshold, L_M, a lower threshold L_R, such as $0 \leqslant L_R < L_M$, and a reserved bandwidth, BW. Additionally, the priority of a queue shaped by BLS can vary between a high and a low value. The low value is usually below the lowest priority of unshaped traffic.

BLS is used with a strict priority scheduler, where BLS modifies the priority seen by the SP depending on a credit counter. Hence, depending on the priority value, the shaped frames can be blocked or not by other classes. However, no matter the state of the credit, if a frame is the first of the queue with the highest priority among the eligible queues, then it will be transmitted. Thus, contrary to CBS, the BLS is a non-blocking shaper, which is a large improvement of the predictability guarantees.

The priority change feature enables the BLS to reserve bandwidth for the shaped queue. This fact induces a low implementation complexity; and also improves fairness in comparison to SP, since it limits the bandwidth available to the shaped queue.

A. Discussion

In this section, we assess the pros and cons of the different scheduling strategies vs the four avionics requirements and challenges, to select the most promising ones:

- **predictability:** thanks to the leaky bucket shapers in the AFDX end-systems and the policers in the switches, all the presented solutions can achieve the necessary determinism and isolation. However, AVB/CBS sometimes blocks frames when the transmission link is free, causing unnecessary delays;
- **modularity:** the solutions compliant with event-triggered paradigm, i.e., NP-SP, DRR, CBS, UBS and BLS, better fulfill the modularity criterion, contrary to time-triggered solutions like TAS and PS;
- **fairness:** as aforementioned, there are four solutions fulfilling the fairness constraint: DRR, CBS, PS and BLS;
- **Complexity:** time-triggered solutions like TAS and PS necessitate the implementation of a complex time synchronisation and induce high complexity; Whereas, CBS, BLS and UBS can be used independently from the synchronisation aspect of AVB and TSN. Nevertheless, UBS induces higher complexity.

The considered solutions vs the main avionics requirements and challenges are illustrated in Table I. Hence, the most promising solutions in the avionics context are DRR and TSN/BLS. The quantitative analysis of these scheduling strategies performance will be conducted, with reference to the already specified solution in the AFDX standard NP-SP scheduling strategy.

Solutions	references	Requirements			
NP-SP	[18]	✓✓	✓✓	X	✓✓
GPS/DRR	[9]	✓✓	✓✓	✓✓	✓
AVB/CBS	[2]	✓	✓✓	✓✓	✓✓
TSN/TAS	[15]	✓✓	X	X	X
TSN/PS	[14]	✓✓	X	✓	X
TSN/BLS	[8]	✓✓	✓✓	✓✓	✓✓
TSN/UBS	[13]	✓✓	✓✓	✓✓	X
Avionics requirements and challenges		Predictability	Modularity	Fairness	Complexity

TABLE I: Existing solutions vs avionics requirements and challenges

IV. Quantitative Analysis of Different Scheduling Strategies

In this section, we conduct performance analysis of the most promising scheduling strategies (BLS and DRR) when incorporated in the AFDX, to evaluate their efficiency to support heterogeneous traffic profiles, in comparison to the current AFDX solution (implementing SP scheduler). First, we describe our representative avionics case study and the testing scenarios. Afterwards, we assess the timing performance and complexity of the selected solutions, in comparison to the current AFDX.

A. Avionics Case Study

Our case study is a representative avionics communication architecture of the A380, based on a 1-Gigabit AFDX[1] backbone network, which consists of 4 switches and 64 end-systems as shown in Fig. 2 (a). The different traffic profiles generated by each end-system are described in Tab. II. Each traffic class $j \in \{SCT, RC, BE\}$ is characterized by $(MFS_j, BAG_j, Deadline_j)$. Figure 2 (b) shows the traffic communication patterns between the source and the final destinations of a given flow. Each circulating traffic flow on the backbone network is a multicast flow with 16 destinations, and crosses two successive switches before reaching its final destinations. The first switch in the path receives traffic from 16 end-systems to forward it in a multicast way to its two neighboring switches. Afterwards, the second switch in the path, which receives traffic from the two predecessor switches, forwards the traffic in its turn to the final end-system.

The main considered performance metrics are:
(i) The **maximum utilisation rate** of each traffic class, that can be sent on the extended AFDX architecture while respecting the schedulability condition. This metric enables the scalability analysis of the extended AFDX with the new scheduling strategies BLS and DRR, in comparison with the current one.
(ii) The **delay bounds** of SCT and RC classes to prove the predictability of the extended AFDX and analyse its impact on the system timing performance, in comparison with the current AFDX implementing SP. It is worth noting that since the BE does not have a deadline, and its largest impact on the other priorities is the transmission time of a maximum sized frame, then the timing performance of this class is not detailed herein. The delay bounds are computed based on Network calculus [10], and particularly the proved results in [5] for BLS and [3] for DRR.
(iii) The **computation time** to tune the parameters of each scheduling strategy to improve as much as possible the system performance when using the tuning methods described in [6].

Priority	Traffic Class	MFS (Bytes)	BAG (ms)	Deadline (ms)
High	SCT	64	2	2
Medium	RC	320	2	2
Low	BE	1024	8	none

TABLE II: Avionics flow Characteristics

The testing scenarios are described in Table III. As it can be noticed, the principle of scenario 1 (resp. scenario 2) is to fix the utilisation rate of RC class UR_{RC} (resp. SCT class UR_{SCT}) at 20% and vary the SCT (resp. RC) utilisation rate to assess the impact of increasing network congestion on the timing performance. The variation of the utilisation rate of a class j is obtained through increasing the number of generated traffic flows within each end-system, n_j^{es}. Thus, the maximum utilisation rate is equal to $UR_j(\%) = \frac{C_j}{C}$ with C_j the capacity used in the bottleneck by the aggregate traffic of class $j \in \{RC, SCT\}$, $C_j = 16 \cdot n_j^{es} \cdot \frac{MFS_j}{BAG_j}$, and C the transmission capacity of the network (1Gbit/s).

[1]The 1-Gigabit version of the AFDX is under specification.

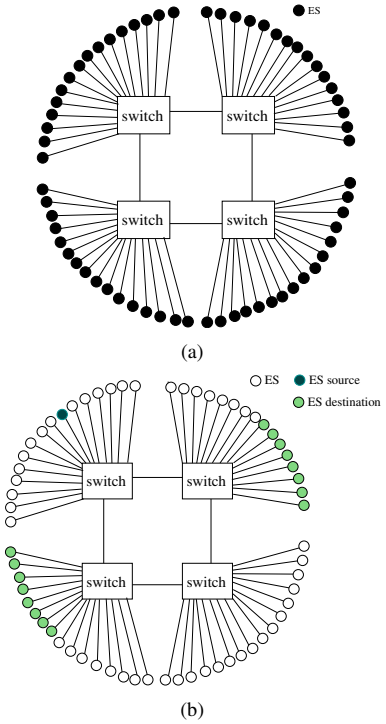


Fig. 2: Representative AFDX network: (a) Architecture; (b) Traffic Communication Patterns

Scenarios	Scenario 1	Scenario 2
$(UR_{RC}; UR_{SCT})(\%)$	$(20; [1..80])$	$([1..80]; 20)$
$(n^{es}_{RC}; n^{es}_{SCT})$	$(10; [1:4:110])$	$([1:2:39]; 47)$

TABLE III: Testing Scenarios 1 and 2

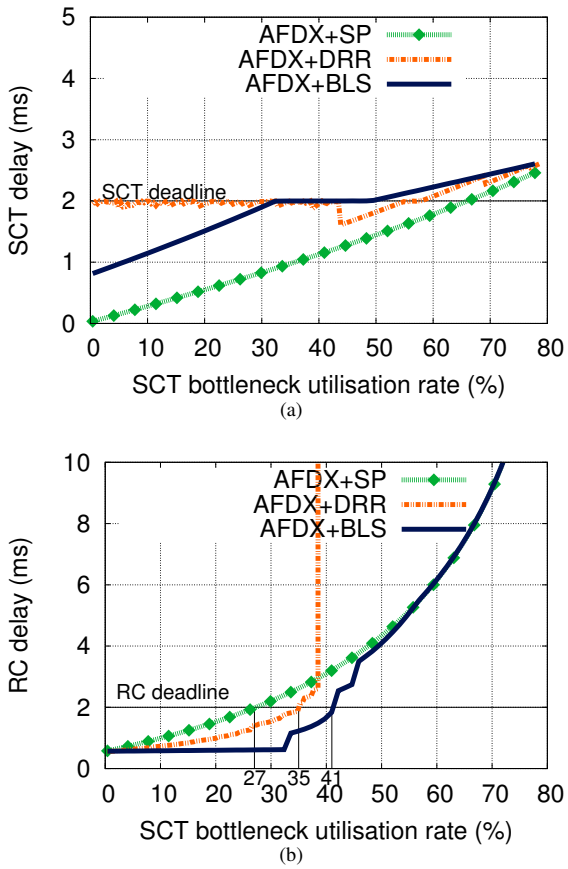


Fig. 3: Testing Scenario 1 Results: (a) SCT delay bounds; (b) RC delay bounds

B. Numerical Results

The results of scenarios 1 and 2 are illustrated in Fig.3 and Fig.4, respectively.

First, concerning the maximum bottleneck utilisation rates:

- in Figure 3, we note that the maximum bottleneck SCT utilisation rate is 27% with the current AFDX (AFDX+SP), 35% with DRR-compliant AFDX (AFDX+DRR) and 41% with the extended AFDX incorporating BLS (AFDX+BLS).
- in Figure 4, the maximum bottleneck RC utilisation rate is 33% with SP, 38% with DRR and 41% with BLS.

Hence, incorporating BLS in the AFDX improves the maximum utilisation rate of RC, compared to both SP (up to 24%) and DRR (up to 17%).

Secondly, concerning timing performance and delay bounds, in Figure 3(b), the RC delay bounds with BLS are lower than the delay bounds with either DRR and SP. In particular, the BLS improves the RC delay bound up to 77% compared to the current AFDX with SP, and up to 73% compared to DRR-compliant AFDX (when the SCT and RC deadlines are fulfilled). The same behaviour is visible in Figure 4(b): the BLS improves the RC delay bounds up to 89% compared to SP, and up to 38% compared to DRR (when the deadlines are fulfilled).

The improvements of the RC delay bounds and schedulability with the BLS and DRR scheduling strategies in AFDX, in reference to the current AFDX (SP), are illustrated in Table IV. We have also computed the computation times to tune the parameters of BLS and DRR to achieve the best performance.

First, we can see that the BLS and DRR improve both the RC delay bounds and the maximum utilisation rates of SCT and RC, compared to SP. We note that the positive impact is much stronger under the BLS than under the DRR. Moreover, we can see that the computation time is multiplied up to 6 times under DRR, in reference to BLS.

Scheduler	improvement compared to SP(%)				computation times (s)	
	maximum RC delay at		maximum			
	$UR^{bn}_{SCT} = 33\%$	$UR^{bn}_{RC} = 28\%$	UR^{bn}_{SCT}	UR^{bn}_{RC}	SCT	RC
BLS	18	22	33	21	57	9
DRR	18	16	26	15	395	58

TABLE IV: Comparing Scheduling Strategies

From these scenarios, we can conclude that with an accurate parameter tuning, the extended AFDX implementing BLS has a large positive impact on both SCT and RC, compared to the current AFDX implementing the SP scheduler or DRR-compliant AFDX, while inducing low complexity.

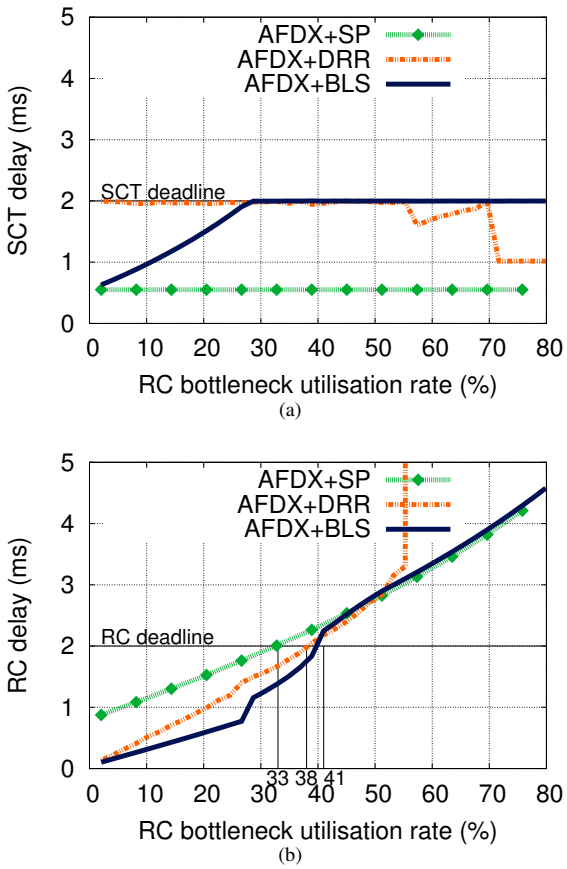


Fig. 4: Testing Scenario 2 Results: (a) SCT delay bounds; (b) RC delay bounds

V. Conclusions

In this paper, we have assessed the most relevant existing scheduling strategies vs the main avionics requirements, to support heterogeneous applications on the AFDX network. Afterwards, we have conducted a quantitative performance analysis of the most promising solutions, i.e., BLS and DRR, in reference with the current one (SP) through a representative avionics case study. Results show the noticeable performance enhancement of the current AFDX traffic (RC) in presence of the highest priority one (SCT) under BLS, with reference to the current AFDX (SP) and DRR, while keeping a low complexity.

References

[1] Airlines Electronic Engineering Committee. Aircraft Data Network Part 7, Avionics Full Duplex Switched Ethernet (AFDX) Network, ARINC Specification 664. Aeronautical Radio, 2002.

[2] Unmesh D Bordoloi, Amir Aminifar, Petru Eles, and Zebo Peng. Schedulability analysis of ethernet avb switches. In *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2014 IEEE 20th International Conference on*, pages 1–10. IEEE, 2014.

[3] Marc Boyer, Giovanni Stea, and William Mangoua Sofack. Deficit Round Robin with network calculus. In *Performance Evaluation Methodologies and Tools (VALUETOOLS)*, 2012.

[4] Alan Demers, Srinivasan Keshav, and Scott Shenker. Analysis and simulation of a fair queueing algorithm. In *ACM SIGCOMM Computer Communication Review*, volume 19. ACM, 1989.

[5] Anaïs Finzi, Ahlem Mifdaoui, Emmanuel Lochin, and Fabrice Frances. Network Calculus-based Timing Analysis of AFDX Networks with Strict Priority and TSN/BLS Shapers. In *SIES*, 2018.

[6] Anaïs Finzi, Ahlem Mifdaoui, Emmanuel Lochin, and Fabrice Frances. Performance Enhancement of Extended AFDX via Bandwidth Reservation for TSN/BLS Shapers. In *RTN workshop in conjunction with ECRTS*, 2018.

[7] Christian M. Fuchs, Advisors Stefan Schneele, and Er Klein. The evolution of avionics networks from arinc 429 to afdx. In *In Proceedings of the Seminars Future Internet (FI), Innovative Internet Technologies and Mobile Communication (IITM) and Aerospace Networks (AN)*, 2012.

[8] Franz-Josef Gotz. Traffic Shaper for Control Data Traffic (CDT)–IEEE 802 AVB Meeting. http://www.ieee802.org/1/files/public/docs2012/new-goetz-CtrDataScheduler-0712-v1.pdf, 2012.

[9] Yu Hua and Xue Liu. Scheduling design and analysis for end-to-end heterogeneous flows in an avionics network. In *INFOCOM, 2011 Proceedings IEEE*, pages 2417–2425. IEEE, 2011.

[10] J.Y. Le Boudec and P. Thiran. *Network calculus: a theory of deterministic queuing systems for the internet*. Springer-Verlag, 2001.

[11] Hyung-Taek Lim, Daniel Herrscher, Martin Johannes Waltl, and Firas Chaari. Performance analysis of the ieee 802.1 ethernet audio/video bridging standard. In *Proceedings of the 5th International ICST Conference on Simulation Tools and Techniques*, 2012.

[12] R. Bosch GmbH. CAN specification Version 2,0. Technical report, 1991.

[13] Johannes Specht and Soheil Samii. Urgency-based scheduler for time-sensitive switched ethernet networks. In *ECRTS*, 2016.

[14] Michael Johas Teener. Back to the future:using TAS and pre-emption for deterministic distributed delays–IEEE 802.1 AVB TG Meeting, San Antonio. http://www.ieee802.org/1/files/public/docs2012/new-avb-mjt-back-to-the-future-1112-v01.pdf, 2012.

[15] Daniel Thiele, Rolf Ernst, and Jonas Diemer. Formal worst-case timing analysis of Ethernet TSN's time-aware and peristaltic shapers. In *VNC*. IEEE, 2015.

[16] TSN Task Group. TSN Specifications. http://www.ieee802.org/1/pages/tsn.html.

[17] C. Watkins and R. Walter. Transitioning From Federated Avionics Architectures To integrated Modular Avionics. 26th Digital Avionics Systems Conference (DASC), 2008.

[18] Adam Wierman and Mor Harchol-Balter. Classifying scheduling policies with respect to unfairness in an m/gi/1. In *ACM SIGMETRICS Performance Evaluation Review*, volume 31, pages 238–249. ACM, 2003.

[19] Tianran ZHOU and Xiong Huagang. Design of energy-efficient hierarchical scheduling for integrated modular avionics systems. *Chinese Journal of Aeronautics*, 2012.

EcoCAM: A Power-saving and Memory Efficient Packet Classification Scheme in TCAMs

Chonghui Ning[†], Wenjun Li[†], Xinwei Liu[†], Ting Huang[†], Wenxia Le[§] and Hui Li[†*]

[†]School of Electronic and Computer Engineering, Peking University, China, [§]Network Energy Department, Huawei, China

{chonghui_ning, wenjunli, lxw0724, huangting53}@pku.edu.cn, lewenxia@huawei.com, lih64@pkusz.edu.cn

***Abstract*—Packet classification is an indispensable yet challenging functionality of real-time data network. Due to the high-speed requirement of network devices, hardware using TCAMs has been the dominant implementation of packet classification in industry. Despite its capability for line-speed queries, TCAM is very power hungry and capacity inefficient. Although SmartPC, a recent proposed energy-efficient TCAM scheme, was recommended to reduce power consumption by constructing a pre-classifier to activate TCAM blocks selectively, its bottom-up based expansion mechanism generated a large number of general rules and memory holes, leading to significant memory overhead. In this paper, we propose EcoCAM, a combination technique of decision-tree and TCAM, which exploits the potentiality of algorithmic and architectural packet classification. To eliminate rule replication, we employ the optimized FiCuts in CutSplit to build decision-tree for the separated subset with similar characteristics. Since the capacity of TCAM block and the number of rules in tree nodes are finite, we construct the count-based tree corresponded to TCAM entries. Finally, using a top-down approach to traverse the tree and build a pre-classifier brings power and storage efficiency. Experimental results show that our algorithm achieves 97.8% power reduction and 5.3% storage overhead on average.**

***Keywords*—Packet Classification, TCAM, Decision Tree, Power Reduction, Memory Efficiency**

I. INTRODUCTION

Packet classification is the crucial technique of modern network routers that provides various network services, such as access control, securities, traffic monitoring, quality of service (QoS) and virtual private network (VPNs). The roles of packet classification is to find out a matching rule with the highest priority from a predefined classifier for each incoming packet and take an action to the packets. Since high-speed packet classification is the bottleneck of real-time data network, this problem has drawn a lot of researchers' attention in the past two decades.

There are many packet classification techniques proposed, which can typically be divided into two patterns: algorithmic and architectural[1][2][3]. Algorithmic solutions, such as decision-tree[4][5][6][7][8][9], decomposition[10][11][12] and tuple space[13][14][15], have drawn wide-spread attention because of their flexibility and scalability. Among them, decision-tree is considered the most promising solution due to the requirements of multi-field packet classification and feasibility of combining with other techniques. As far as we know, a plethora of decision-tree algorithms has been exhibited. Using a geometric view, decision-tree partitions the searching space into multiple subspaces. There are two major partitioning mechanisms to construct a decision-tree: *equal-size cutting* and *equal-dense splitting*. HiCuts[4] and HyperCuts[5], the cutting-based approaches, take advantage of local optimizations to cut the searching space, but they are not applicable for large rule sets because of sharply growing rule replication. After that, EffiCuts[7] employs the subset partitioning techniques to achieve significant reduction of storage overhead, however, a K-field classifier may be separated into 2^K subsets at most. On the other hand, HyperSplit[6], an outstanding splitting method, divides the searching space into varying subspaces containing a similar number of rules. The state-of-the-art technique, CutSplit[9], exploits the combination of cutting and splitting schemes and realizes less storage overhead and fast updates. Although above approaches have made great achievements in performance and storage, line-speed requirement is still a challenging problem, which can be lightly implemented by hardware-based methods.

Ternary Content Addressable Memories (TCAMs) based techniques are the most popular architectural approaches in industry. TCAMs accomplish line-speed packet classification by paralleling lookups the rule set in a single pass. However, they are not only expensive, but also area-inefficient and power-hungry. To address these problems, numerous methods have been put forward, such as *classifier minimization*[16][17], *range encoding*[18][19][20], *circuit modification*[21] and *pre-classifier*[22][23][24][25][26][27]. Among them, the structure of pre-classifier enables TCAM to selectively activate a small number of TCAM blocks by establishing index TCAM, resulting in the reduction of power consumption. Based on this characteristic, Extended TCAM[21] employs circuit modification to convert range to prefix and constructs corresponding index entries to reduce energy consumption. But circuit modification is undesirable since TCAM is widely used in industry. Recently, more efficient algorithms to reduce power consumption have been proposed, such as TreeCAM[23], which combines decision-tree and TCAM to attain fast updates and less energy consumption but brings about extra storage overhead. SmartPC[24] employs a bottom-up algorithm to construct a pre-classifier by grouping rules in the same area into TCAM blocks. However, such a local optimal pattern generates plenty of general rules and holes in TCAM blocks. The state-of-the-art technique, GreenTCAM[26], builds the pre-classifier by projecting rules into different dimensions to select the range with similar numbers of rules, which improves energy and memory utilization compared with SmartPC. But it introduces an additional SRAM structure and brings about rule replication.

*Corresponding author Hui Li is also with the Shenzhen Key Lab of Information Theory & Future Network Architecture, Future Network PKU Lab of National Major Research Infrastructure, Shenzhen Engineering Lab of Converged Networking Technology, Huawei & PKU Jointly Engineering Lab of Future Network Based on SDN and the PKU Institute of Big Data Technology, Shenzhen Graduate School, Peking University, China. Chonghui Ning, Xinwei Liu and Ting Huang do this work under the guidance of the second corresponding author: Wenjun Li.

In this paper, we propose the eco-friendly TCAMs (EcoCAM), an efficient scheme combining decision-tree and TCAMs, which enhances power reduction and storage utilization simultaneously. The combination of algorithmic packet classification (using decision-tree) and architectural packet classification (using TCAM) techniques is beneficial for taking advantage of the relationship among rules to reduce energy consumption and improve storage efficiency from a global perspective. At first, we partition the rule set into subsets in which the rules have similar characteristics. Since the capacity of every TCAM block is finite, we use the subset to construct a count-based tree by mapping decision-tree to correspond to the entries of every TCAM block. Finally, we employ a greedy-based heuristic algorithm to traverse the count-based tree top-down, arrange the tree nodes into TCAM blocks.

Using the rule sets up to 100k entries generated by ClassBench[28], we evaluate our algorithm performance in power reduction and storage overhead. Compared with SmartPC and GreenTCAM, EcoCAM further enhances power reduction with an average of 97.8% and achieves merely 5.3% storage overhead on average. Moreover, we implement SmartPC for an adequate comparison.

The rest of the paper is organized as follows. In Section II, we briefly introduce the problem of packet classification and summarize the related work about decision-tree and TCAM-based solutions. Section III presents the technical details of EcoCAM. Section IV provides experimental results. Finally, Section V draws conclusion of this paper.

II. BACKGROUND AND RELATED WORK

Packet classification is performed to find a matching rule with the highest priority from a packet classifier. A classifier contains a set of rules with a fixed number. Typically, a rule R is consist of 5-tuple and an action to be taken when a packet matches. The 5-tuple includes source IP address (SA), destination IP address (DA), source port number (SP), destination port number (DP) and the protocol (Prot). Rule field values could be exact value, prefix or range, which means there are three kinds of matches: exact match, prefix match or range match. When a packet arrives, packet classification needs to compare the header of the packet with a predefined classifier to find out the best match and take action. Although a plethora of algorithmic and architectural approaches has been proposed, packet classification remains a challenging problem. Among them, decision-tree and TCAM are the most representative software-based and hardware-based solutions.

A. Decision-tree based schemes

In essence, decision-tree schemes make use of the geometric perspectives (as shown in Figure 1) of the packet classification problem to reduce the searching space by constructing a tree. The whole searching space containing all rules is covered by the root node. With the increase of depths, the searching space for tree nodes narrows. At the end nodes of decision-tree, linear search algorithm is efficient enough in a smaller number of rules. EffiCuts[7], as a famous cutting-based scheme, proposes a significant scheme based on rule sets partitioning to reduce rule replication dramatically. However, A K-field classifier could be separated into at most 2^K subsets, leading to lots of memory access. Later, HybridCuts[8] realizes less memory accesses

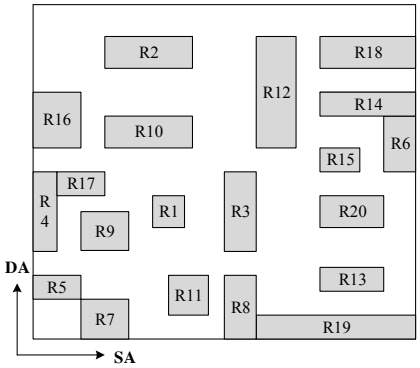


Figure 1. Geometric view of *small rules*

compared with EffiCuts by adopting a single individual field (SA or DA) to separate rules rather than all K fields; but the worst-case performance is unbounded because of the utilization of HyperCuts[5]. What's more, HybridCuts proposes a cutting-based algorithm Fixed intelligent Cutting (FiCuts) which brings about no rule replication in cutting phase. As an improvement, the state-of-the-art technique CutSplit[9] optimizes FiCuts to support multiple field cutting with the same effects as HybridCuts and employs an ingenious method to incorporate cutting and splitting, achieves less storage overhead and fast updates. Even so, line-speed requirement is still the primary bottleneck of decision-tree based solutions.

B. TCAM based schemes

TCAMs have become the de facto standard device of hardware-based packet classification because of the requirements of high performance routers and switches. However, high power consumption restricts its developments. To solve this problem, TCAM vendors create a partitioning mechanism that only activates a small number of TCAM blocks associated with the input bits, called *pre-classifier*. A typical pre-classifier architecture is shown in Figure 2.

Extended TCAM[21] employs circuit modification to support range transformation and construct index entries, only segmental TCAM blocks associated with the index will process when packets arrive. But circuit modification is almost impossible to commercialize limited by existing market volume. CoolCAMs[22], a TCAM-based IP lookup approach, exploits the features of IP prefix tree based on longest prefix match to build index entries by multiple sub-tree splitting, which achieves a huge number of power reduction and memory utilization. However, the narrow application of IP lookup constraints its developments.

Lately, more efficient TCAM-based approaches have been proposed. For example, TreeCAM[23], a typical combination technique of decision-tree and TCAMs, builds decision-tree in which the leaf nodes containing the number

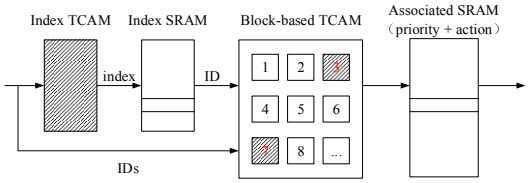


Figure 2. Architecture of TCAM pre-classifier

of rules satisfied a TCAM sub-array and stores these nodes in nearby area of TCAMs, resulting in fast updates and energy reduction. But it brings rule replication and memory inefficiency. SmartPC[24] proposes a two-stage algorithm based on bottom-up approach: at first, a packet is classified by a pre-classifier that identified the TCAM block; only the identified block along with a few general blocks are activated and searched in parallel. Although its power consumption is reduced 88% on average, general rules produced by SmartPC have become primary restriction of energy reduction, which have to be searched for each incoming packet. Besides, SmartPC may produce holes in blocks, which waste a lot of TCAM capacity. As an improvement, GreenTCAM[26] attempts to reduce rule replication by partitioning rules into subsets depending on SA and DA field, then projects rules to attain a smaller subset with a number of rules nearly to the entries of TCAM block. Although GreenTCAM has better performances in power consumption and storage overhead, it brings about rule replication and introduces an extra SRAM structure to configure index TCAM since TCAM does not support range storage.

III. THE PROPOSED ALGORITHM

In this section, we first briefly summarize the challenges of combining decision-tree and TCAM and then put forward some ideas. On this basis, we try to construct a count-based tree via mapping decision-tree. Furthermore, we propose two cutting algorithms based on the count-tree to construct pre-classifier.

A. Challenges & ideas

First, we present some definitions to describe our algorithm adequately, as follows:

- *nrules*: the rule number of tree node.
- *binth*: the threshold of leaf node.
- *block-size*: the capacity of one TCAM block.
- *general block*, *general rule* and *special block*: as mentioned in SmartPC.

Although TCAM is the most popular hardware-based packet classification approach in industry, power consumption has become the primary constraint of its development. When it comes to designing a power-saving TCAM-based solution, there are two considerable points: *general rules* and storage efficiency. *General rules* are inevitable sometimes because of the existence of "*" (wildcard), but we can try to lessen them as much as possible. On the other hand, rule replication also brings about extra power consumption since a rule may store in more than one TCAM blocks which causes multiple blocks to be activated when a matching packet arrives. Besides, rule replication generates extra storage overhead. Therefore, the challenges of lower power consumption TCAM are *general rules* and memory efficiency.

TABLE I. NRULES DISTRIBUTION OF LEAF NODES (*BINTH*=4)

Rule set	*Nrules range*				
	0-4	**4-8**	**8-16**	**16-32**	**>32**
ACL	91.07%	2.97%	2.60%	2.45%	0.91%
FW	90.89%	5.28%	3.01%	0.37%	0.45%
IPC	93.08%	1.94%	1.71%	2.65%	0.63%

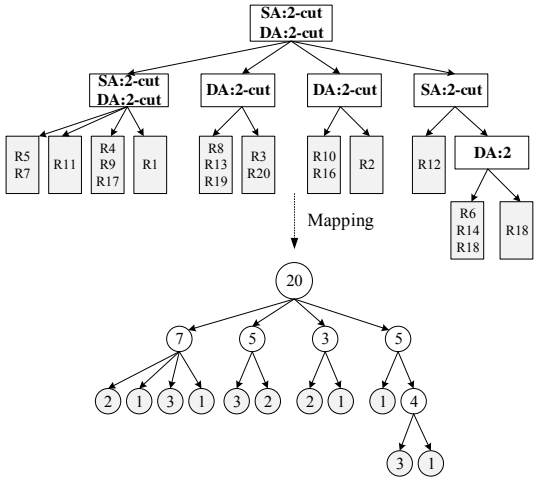


Figure 3. Decision-tree mapping of *small set* with Figure 1 *(binth*=2)

Existing researches present that the real classifiers have some common characteristics and inherent redundancies which can be exploited to reduce rule replication[4][5][7][9]. For example, the address fields source address (SA) and destination address (DA) provide sufficient separability even for multi-field rules. And the distribution of real rules is unbalanced. The state-of-the-art research, CutSplit[9], proves that only a few rules have lager range lengths of fields (called *big rules*) and the vast qualities of rules share a small range lengths of fields (called *small rules*). Lots of rules among small rules have distinctive fields. These features are still valid for SA and DA fields. Therefore, CutSplit proposes that building decision-tree without rule replication in cutting phase is possible and implements the modified FiCuts in HybridCuts. Moreover, *block-size*, as the capacity of one TCAM block, is a finite number which means that storing the tree node whose *nrules* is close to *block-size* into a *special block* to construct a pre-classifier will receive high performance. Next, we will present the details of our design.

B. Count-tree

As mentioned above, the purpose of our algorithm is to find the nodes whose *nrules* is the most nearly to *block-size*. So we propose a count-based decision-tree algorithm to capture the *nrules* of every node, called *count-tree*. First of all, we partition the rule set based on SA and DA and put *big rules* in *general blocks* because they may bring about rule replication and activate in more than one *special blocks*. We then make use of the remaining subsets to construct the count-tree by mapping the decision-tree. Figure 3 shows that how to use the *small set* in Figure 1 to build the count-tree with two-field cutting. The details of count-tree construction are as follows:

- Step 1: **partition rule set.** *1) Big set:* SA and DA are both big fields, *2) SA set* (SA cutting): SA is small field and DA is big field, *3) DA set* (DA cutting): DA is small field and SA is big field and *4) Small set* (two-field cutting): SA and DA are both small field.
- Step 2: **remove big set.** Store *big rules* into *general blocks*.
- Step 3: **construct decision tree.** Based on the optimized FiCuts in CutSplit, we can construct a

decision-tree without rule replication. Besides, we introduce *parent pointers* and *child pointers* that respectively point to their parent nodes and child nodes to achieve rule updates and sub-tree splitting in splitting algorithm mentioned below.

- Step 4: **count-tree mapping.** The implementation of count-tree is simple via only storing *nrules* in every tree node.

As a result, a count-tree without rule replication has been built. Using ClassBench[28], we evaluate the *nrules* distribution of leaf nodes in our algorithm since FiCuts will result in that the *nrules* of leaf nodes may be greater than *binth*. As shown in TABLE I, only a few leaf nodes' *nrules* is greater than *binth*, less than 10%. Among them, less than 1% of the leaf nodes have a large *nrules* than 32 which is far less than *block-size* in practice. Thanks to characteristics of *nrules* distribution, the nodes of the entire tree may exist a node that its rule number is close to *block-size* even equal.

C. Abs-based splitting

Based on above observations, we propose our first algorithm, called ***abs-based splitting***. Building a count-tree with a small *binth* enables that the leaf nodes only include a little rules. Taking advantage of this feature, we introduce another value Δ, the absolute value between *nrules* and *block-size*, as the judgment criteria of the best carving node. Apparently, if Δ gets closer to zero, *nrules* is closer to *block-size*. Therefore, we select the best carving node by sorting Δ in descending order. Even if *nrules* is greater than block-size, Δ is the smallest in all nodes, only storing Δ nrules into *general block* is acceptable for better storage overhead and power reduction.

We present the pseudo code on how to use abs-based splitting to build a pre-classifier in Algorithm 1. This algorithm inputs a count-tree *tree* and a parameter *block-size b*. The entire tree in our algorithm is traversed in level order looking for the best carving node. The best carving node is a node whose Δ is the smallest. Different from SmartPC and CoolCAMs, this up-down approach is global optimal. Every time the best carving node is found, put the rules of this node in *special block* and put relevant index in index TCAM if *nrules* is less than *block-size*. Otherwise, divide the rules into two parts. Among them, the lager one containing *block-size* rules stores in *special block* and updates index TCAM, the other with Δ rules is put in *general block*. Using *parent pointers* and *child pointers*, the entire tree removes all the rules of this node and marks its descendant nodes and itself as NULL to implement sub-tree splitting and no rule replication. At the same time, update the Δ and *nrules* of entire tree. End the loop until the root node includes the number of rules which is less than *block-size*, put it in *special block* and update index TCAM. Figure 4 shows that how to carve out a count-tree using abs-based splitting.

Algorithm 1 abs-based splitting (tree, b)

```
 1: while (root.nrules > b)
 2:     traverse tree with level order;
 3:     find p = the node with smallest Δ
 4:     if (p.nrules > b)
 5:         put Δ rules(p) of p in general block;
 6:         put other b rules(p) in new special block;
 7:         put prefix(p) in index TCAM;
 8:     else
 9:         put rules(p) in new special block;
10:         put prefix(p) in index TCAM;
11:     for each u is descendant of p do
12:         u = NULL;
13:     p = NULL;
14:     for each v is ancestor of p from p to root do
15:         v.nrules = v.nrules - p.nrules;
16:         v. Δ = | v.nrules - b |;
```

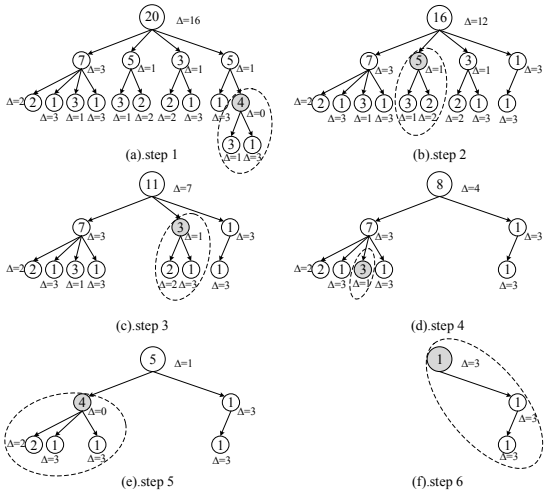


Figure 4. An example of abs-based splitting (*block-size*=4)

D. Level-order splitting

Obviously, the abs-based splitting algorithm will still ineluctably produce *general rules*. As an improvement, we put forward another tree splitting algorithm, called ***level-order splitting***. The *nrules* of leaf nodes are very small as mentioned before. Making use of this property, level-order splitting builds a pre-classifier with multi-prefix to fill up TCAM block as much as possible.

Algorithm 2 level-order splitting (tree, b)

```
 1: while (root.nrules > b)
 2:     traverse tree with level order;
 3:     find p = the node is closest to b but not greater than b;
 4:     let left = b – p.nrules;
 5:     if (left = 0)
 6:         put rules(p) in new special block;
 7:         put prefix(p) in index TCAM;
 8:         for each u is descendant of p do
 9:             u = NULL;
10:         for each v is ancestor of p from root to p do
11:             v.nrules = v.nrules - p.nrules;
12:     else
13:         put rules(p) in new special block(b);
14:         put prefix (p) in index TCAM(i);
15:         for each u is descendant of p do
16:             u = NULL;
17:         for each v is ancestor of p from p to root do
18:             v.nrules = v.nrules - p.nrules;
19:         while ( left > 0)
20:             traverse tree with level order;
21:             find q = the node is closest to b but not greater than left;
22:             left = left – q.nrules;
23:             put rules(q) in old special block(b);
24:             put prefix(p) in old index TCAM(i);
25:             for each u is descendant of q do
26:                 u = NULL;
27:             q = NULL;
29:             for each v is ancestor of q from q to root do
30:                 v.nrules = v.nrules - p.nrules;
```

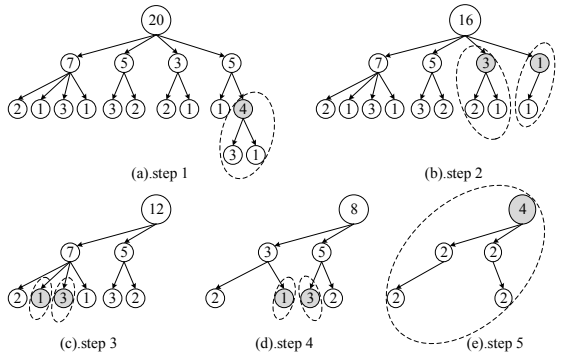


Figure 5. An example of level-order splitting (*block-size*=4)

Similar to abs-based splitting, the input parameters are a count-tree *tree* and the *block-size b*. But extra value Δ is removed because the structure of multi- prefix only needs the nodes whose *nrules* are less than or equal to *block-size*. The pseudo code of level-order spitting is shown in Algorithm 2. First, find the node whose *nrules* is the closest to but not greater than *block-size*. Define *left* to be the difference value between *nrules* and *block-size*. If *left* is zero that means this node fills up one TCAM block, put the rules of this node in s*pecial block* and relevant index in index TCAM. If not, find another node whose *nrules* is the closest to but not greater than *left* and let *left* be the surplus entries of TCAM block. Then judge whether *left* is zero to end the loop and put the rules of these nodes in the same *special block* and relevant index in the same index TCAM. Every time the carving node is found, the entire tree removes all the rules in this node and marks its descendant nodes and itself as NULL with *parent pointers* and *child pointers*. End the loop until the root node contains rules that is less than *block-size*, put it in *special block* and update index. Although the employment of multi-prefix brings extra consumption in pre-classifier, it is negligible compared to the significant decrease in *general rules*. Figure 5 illustrates an example of level-order splitting.

IV. Experiment Results

In this section, we evaluate the performance of EcoCAM compared with SmartPC[24] and GreenTCAM[26]. The rule sets used in our experiments are generated by ClassBench[28]. We choose the 100k rules based on three representative types of rule sets: Access Control List (ACL), Firewall (FW) and IP Chain (IPC). Besides, we implement the classical algorithm SmartPC which is publicly available in [29] as well as our algorithm. Based on existing schemes, we evaluate our algorithm from power reduction and storage overhead.

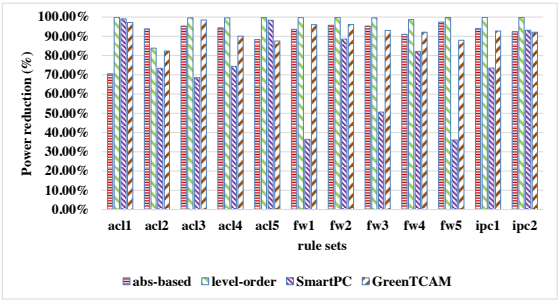


Figure 6. Power reduction for rule sets with block-size 256

TABLE III. General Rules Using Abs-Based Splitting

Rule sets	Block-size				
	128	**256**	**512**	**1024**	**2048**
acl1_100k	26401	28687	840	8845	23901
acl2_100k	2581	5158	3439	13574	20511
acl3_100k	32199	4365	11030	15665	22507
acl4_100k	9046	5298	11981	14192	28556
acl5_100k	25300	11523	0	8347	24702
fw1_100k	12624	4909	15092	22127	10518
fw2_100k	26680	3770	7170	13653	32500
fw3_100k	19102	3370	11677	16368	23703
fw4_100k	6048	6399	17131	21807	7188
fw5_100k	16747	2026	8579	27641	21574
ipc1_100k	23196	5550	8310	15365	32530
ipc2_100k	23713	7569	18084	18283	28056

TABLE II. General Rules Using Level-Order Splitting

Rule sets	Block-size				
	128	**256**	**512**	**1024**	**2048**
acl1_100k	5	5	5	5	5
acl2_100k	14837	13588	106	106	106
acl3_100k	336	336	336	336	336
acl4_100k	373	373	373	373	373
acl5_100k	0	0	0	0	0
fw1_100k	199	199	199	199	199
fw2_100k	3	3	3	3	3
fw3_100k	250	250	250	250	250
fw4_100k	12695	920	920	920	920
fw5_100k	193	193	193	193	193
ipc1_100k	147	147	147	147	147
ipc2_100k	0	0	0	0	0

A. Power reduction

As we discussed earlier, general rules are the main factor that affect power consumption in TCAM-based algorithms. Furthermore, the structure of pre-classifier also brings a little energy consumption. For these purposes, we take account of these two affecting elements to evaluate the performance of power reduction. Since the rule sets of our evaluation is 100k, we select 5 different TCAM *block-size*: 128, 256, 512, 1024 and 2048 representatively. Because of limited space, TABLE II and TABLE III only display the number of general rules of our algorithms: *abs-based splitting* and *level-order splitting*. Figure 6 shows the power reduction of our algorithms with block-size 256 compared with SmartPC and GreenTCAM. On the other hand, the ratio of pre-classifier entries ranges from 0.1% to 1% as the block-size decrease, leading to 0.1% to 1% power consumption.

Clearly, our algorithms achieve enormous results in general rules reduction, almost remove all of them. As a result, the power reduction ratio of abs-based splitting ranges from 65.3% to 98.8%, and the average is 91.1%. Another scheme level-order splitting, as the improved approach, achieves a higher power reduction ranging from 84.6% to 99.5% and with an average of 97.8%. Besides, level-order splitting works better with the increase of block-size. Compared to the average power reduction of 88% in SmartPC and 93.6% in GreenTCAM, we receive an excellent performance.

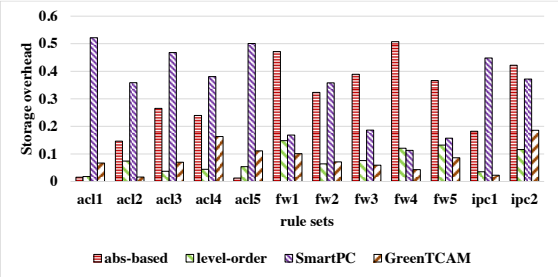


Figure 7. Storage overhead for rule sets with block-size 256

B. Storage overhead

Storage overhead is another significant evaluation factor because of the high price and finite capacity of TCAM. First of all, our algorithm won't produce rule replication since the construction of count-tree employs the optimized FiCuts in CutSplit. Moreover, the top-down approach and building a pre-classifier with multi-prefix will fill up the TCAM blocks as much as possible. Although the abs-based splitting has a large range from 0.5% to 50.7% because of the unstable value Δ and its average is 27.7%. The level-order splitting takes advantage of pre-classifier with multi-prefix to enhance the usage of TCAM block entries, which ranges from 1% to 14.8% and has average storage overhead of 5.3%. GreenTCAM has the similar performance with average 5.6% in storage overhead, but it brings about rule replication and needs extra SRAM. SmartPC, however, suffers from huge inefficient memory due to its bottom-up expanding mechanism. The storage overhead of SmartPC ranges from 10.8% to 61% and has an average of 43.1%. Figure 7 shows the storage overhead of our algorithms, SmartPC and GreenTCAM with block-size 256.

V. CONCLUSION

In this paper, we propose EcoCAM that employs the combination of decision-tree and TCAM to significantly reduce the power consumption and enhance the memory utilization of TCAM. Since the entries of TCAM block is finite, we build a count-based tree by mapping a decision-tree and select appropriate tree nodes to make up the pre-classifier. The construction of decision-tree using the optimized FiCuts in CutSplit removes all rule replication. Taking a top-down approach, tree traversal is global optimal, resulting in higher power reduction and less storage overhead. And the adoption of multi-prefix improves the performance once again. The experimental results show our algorithm achieves power reduction up to 99.5% with an average of 97.8% and the storage overhead is only 5.3% on average.

ACKNOWLEDGMENT

This work is supported by National Key Research & Development Program of China (No. 2017YFB0803204 & 2016YFB0800101), National Natural Science Foundation of China (NSFC) (No. 61671001), Guangdong Key Program (GD2016B030305005) and Shenzhen Research Programs (JSGG20150331101736052, ZDSYS201603311739428, JCYJ20170306092030521). This work is also supported by the Shenzhen Municipal Development and Reform Commission (Disciplinary Development Program for Data Science and Intelligent Computing).

REFERENCES

[1] D. E. Taylor, "*Survey and Taxonomy of Packet Classification Techniques*," ACM Computing Surveys, 37(3):238-275, 2005.

[2] H. J. Chao and B. Liu, "*High performance switches and routers*," John Wiley & Sons, 2007.

[3] C. R. Meiners, A. X. Liu and E. Torng, "*Hardware Based Packet Classification for High Speed Internet Routers*," Springer, 2010.

[4] P. Gupta and N. McKeown, "*Packet Classification using Hierarchical Intelligent Cuttings*," in IEEE Hot Interconnects, 1999.

[5] S. Singh, F. Baboescu, G. Varghese and J. Wang, "*Packet Classification using Multidimensional Cutting*," in ACM SIGCOMM, 2003.

[6] Y. Qi, L. Xu, B. Yang, Y. Xue and J. Li, "*Packet Classification Algorithms: From Theory to Practice*," in IEEE INFOCOM, 2009.

[7] B. Vamanan, G. Voskuilen and T. Vijaykumar, "*EffiCuts: Optimizing Packet Classification for Memory and Throughput*," in ACM SIGCOMM, 2010.

[8] W. Li and X. Li, "*HybridCuts: A Scheme Combining Decomposition and Cutting for Packet Classification*," in IEEE Hot Interconnects, 2013.

[9] W. Li, X. Li, H. Li and G. Xie, "*CutSplit: A Decision-Tree Combining Cutting and Splitting for Scalable Packet Classification*," in IEEE INFOCOM, 2018.

[10] V. Srinivasan, G. Varghese, S. Suri and M. Waldvogel, "*Fast and Scalable Layer Four Switching*," in ACM SIGCOMM, 1998.

[11] T. V. Lakshman and D. Stiliadis, "*High-speed Policy-based Packet Forwarding Using Efficient Multi-dimensional Range Matching*," in ACM SIGCOMM, 1998.

[12] P. Gupta and N. McKeown, "*Packet Classification on Multiple Fields*," in ACM SIGCOMM, 1999.

[13] V. Srinivasan, S. Suri and G. Varghese, "*Packet Classification using Tuple Space Search*," in ACM SIGCOMM, 1999.

[14] T. Yang, G. Xie, Y. Li, Q. Fu, A. X. Liu, Q. Li and L. Mathy, "*Guarantee IP Lookup Performance with FIB Explosion*," in ACM SIGCOMM, 2014.

[15] T. Yang, A. X. Liu, Y. Shen, Q. Fu, D. Li and X. Li, "*Fast OpenFlow Table Lookup with Fast Update*," in IEEE INFOCOM, 2018.

[16] A. X. Liu, C. R. Meiners and Y. Zhou, "*All-Match Based Complete Redundancy Removal for Packet Classifiers in TCAMs*," in IEEE INFOCOM, 2008.

[17] A. X. Liu, C. R. Meiners and E. Torng, "*TCAM Razor: A Systematic Approach Towards Minimizing Packet Classifiers in TCAMs*," IEEE/ACM Transactions on Networking, 18(2):490-500, 2010.

[18] H. Liu, "*Efficient Mapping of Range Classifier into Ternary-CAM*," in IEEE Hot Interconnects, 2002.

[19] K. Lakshminarayanan, A. Rangarajan, and S. Venkatachary, "*Algorithms for Advanced Packet Classification with Ternary CAMs*," in ACM SIGCOMM, 2005.

[20] H. Che, Z. Wang, K. Zheng and B. Liu, "*DRES: Dynamic Range Encoding Scheme for TCAM Coprocessors*," IEEE Transactions on Computers, 57(7):902-915, 2008.

[21] E. Spitznagel, D. E. Taylor and J. Turner, "*Packet Classification Using Extended TCAMs*," in IEEE ICNP, 2003.

[22] F. Zane, G. Narlikar and A. Basu, "*CoolCAMs: Power-Efficient TCAMs for Forwarding Engines*," in IEEE INFOCOM, 2003.

[23] B.Vamanan and T.Vijaykumar, "*TreeCAM: Decoupling Updates and Lookups in Packet Classification*," in ACM CoNEXT, 2011.

[24] Y. Ma and S. Banerjee, "*A Smart Pre-Classifier to Reduce Power Consumption of TCAMs for Multi-dimensional Packet Classification*," in ACM SIGCOMM, 2012.

[25] Z. Ruan, X. Li and W. Li, "*An Energy-efficient TCAM-based Packet Classification with Decision-tree Mapping*," in IEEE TENCON, 2013.

[26] X. Li, Y. Lin and W. Li, "*GreenTCAM: A Memory- and Energy-efficient TCAM-based Packet Classification*," in IEEE ICNC, 2016.

[27] W. Li, X. Li and H. Li, "*MEET-IP: Memory and Energy Efficient TCAM-based IP Lookup*," in IEEE ICCCN, 2017.

[28] D. E. Taylor and J. S. Turner, "*Classbench: A Packet Classification Benchmark*," in IEEE INFOCOM, 2005.

[29] https://github.com/nch30/EcoCAM.

Diting: A Real-time Distributed Feature Serving System for Machine Learning

Meng Wan, Chongxin Deng, Siyu Yu
Vectorlab, JD Finance,
Beijing 100176, China,
wanmeng@jd.com, dengchongxin@jd.com, yusiyu@jd.

***Abstract*—4G networks dramatically boost the speeds and coverage of networks, and there are mounting mobile data produced by data sources of 4G. The generated data can be applied into preventing financial criminal activities, for example, account leakage risks can be prevented with such data instantly, which demands short time delay to predict the fraud, at most within 50 ms. Furthermore, the lower latency, higher data capacity of forthcoming 5G networks will create a new platform for the delivery of services wherever the 5G network exists, paving a way for artificial intelligence-based banking services. Real-time prediction services and data analytics services that can be integrated into business applications have become eagerly demanded in recent years. However, most machine learning and deep learning platforms only provide offline model training&test and model predictions without real-time feature processing. Diting is an easy-to-use distributed intelligent machine learning platform, providing offline model training and low-latency feature serving for predictions in real time. Diting offers incremental feature computing, combines technologies of resource scheduling, rule engines, and Remote Procedure Call (RPC), and builds a real-time distributed computing framework, thus offers low-latency end-to-end prediction services. Diting is also a collaborative, drag-and-drop and virtualization tool. Domain expert users without any programming knowledge can quickly fulfill their business logic. Using real production data for over two months, we show that Diting tremendously improves productivity, and bridges the gap between offline and real-time feature engineering.**

Keywords: Real-time Computing, Incremental Aggregation, Distributed Systems, Feature Engineering.

I. Introduction

Since the first 4G networks were launched, e-commerce has undergone an extensive proliferation in the past decade, now over 80% of online shopping is running on 4G networks in Jingdong (JD), the world's third-largest internet company by revenue and China's largest online retailer. JD Finance, the wealth management platform of JD, was established to give individuals and businesses quick, easy and convenient access to the financial service. The pervasion of 5G networks could revolutionize the e-commerce and the finance industry by Virtual Reality (VR), Augmented Reality (AR) technologies and predictive machine learning-based services. These services generally collect a user's behavioral data in real time to recommend location-based financial advice, such as suggesting new ways to save at the retail store and offering more precise and valuable wealth management consultation.

Nowadays, machine learning (ML) has come to play an integral role in many stages of the financial ecosystem, from assessing risks to granting loans, to the handling of bad assets, to fraudster detection. ML models are required to be trained with a wide range of features. Solving a ML problem often involves the following steps: gathering data, cleaning data (ETL), feature engineering (FE), model training, testing and prediction [1]. ETL stands for Extract, Transform, Load, and it aims to provide clean data before FE. In Diting, ETL is incorporated into FE. In order to build a model that can make the best prediction or recommendation, the first thing one needs to do is to generate appropriate features from the datasets; better features are the deciding factor of the performance of a system.

With advancements in technology, more and more predictions are expected to be done in real time [2]. Identifying fraudulent purchases in a particular time frame in e-commerce scenarios can prevent losses of millions of dollars per year. However, FE of traditional artificial intelligence platforms often focused on a batch or offline FE. It would take a team of data engineers additional months to develop a manual pipeline of real-time FE. To the best of our knowledge, there is no precise report concerning systems that automatically provide feature serving in real time using the same logic as offline feature cleansing. *Real-time feature serving* is defined as providing the feature processing service for real-time ML prediction, bringing feature data over the network in real time.

Rather than force the session to process the entire source data from scratch, incremental computing is generally adopted to do the real-time computing. Intuitively, incremental FE computation can be implemented using *dynamic/incremental algorithms* methods, where programmers need to design the logic that maintains the value of a function over an evolving set, for example, tracking the maximum amount of payment that changes with insertions or deletions of incoming tuples [3]. Nevertheless, after a large body of prior research in the incremental algorithms, we found that although they are efficient, they are usually complex to realize even for simple *fully dynamic* questions like dynamic connectivity [4], [5], [6]. Therefore, we resort to *incremental sliding window analytics* [6] to tackle real-time computation, and better yet, certain characteristics of sliding windows are exploited so that the time consumption of a single incoming tuple is not dependent on the size of the whole historical data but rather commensurate with the number of intermediate results.

In summary, the main contributions of this paper are ab-

stracted as follows:

- **an easy-to-use distributed ML framework and analytics:** Machine learning can be hard to grasp for most people, especially those who are from a non-programming background. Diting is a drag-and-drop tool that automates machine learning and does not require any coding. It's a visualized modeling tool that lets users build custom models, train them, tune them by adjusting parameters, and deploy them in applications. The model is built in the distributed cluster, its performance is guaranteed.
- **support for custom-built end-to-end FE pipelines:** Diting eliminates most of the draining operations related to preparing features for ML, ranging from cleansing, normalization, feature extraction to feature transformation.
- **offer a unified framework for real-time and offline FE and model prediction service:** FE is the process of constructing new features from existing data to train an ML model. Diting removes human-labor processes for users, provides an automatic offline-to-real-time projection of FE & ML pipelines.

II. RELATED WORK

Spark streaming [7] processes data streams in mini-batches, where each mini-batch collects a set of events that arrived over the batch period. The minimum interval at which data received is recommended to be 50 ms. While we require each and every record is processed as it arrives, and the maximum latency is 50 ms.

Storm uses topologies to do real-time computation. For a Storm topology, the user has to specify the amount of resources, and the resources is not shared between topologies, therefore the maximum amount of runnable topologies is finite. Whereas in Diting, real-time DAGs can be dynamically added or removed as rules, are evenly distributed in the cluster by dynamic scheduling, so the resources of CPU and memory of the cluster isn't occupied exclusively by any DAG, there is no limitation of the number of the DAGs. Once the average loads of CPUs or the heap size per worker (JVM) is too high on a cluster, new servers can be added to the cluster. Besides, Storm still requires manual programming, whilst coding is removed from Diting system by configuring rules.

III. IMPLEMENTATION & DESIGN

Since the problem of *Real-time feature serving* is generic, any other organizations can draw on the experience we describe here no matter what industries they involve in as long as unifying offline and real-time FE is business-critical.

A. Highlights of Our Approach

Incremental Aggregation for FE. Incremental aggregation aims to tackle the problem of real-time computing of a limited set of aggregate operators such as $mean$, max, min, sum, $count$, .etc, and a large number of complex aggregates of the above basic operators. Besides, one predictive computation is customized for certain business scenarios. If a customer demands more features, then accordingly the pipelines could be adjusted to it.

We first consider some fundamental operators in database systems and Data Stream Management Systems (DSMS) [8], [9], [10], [11]. They are categorized into *distributive* (e.g., max, min, sum, and count) aggregates and *algebraic* (e.g., avg, std, trend, skew) aggregates, and some *holistic* aggregates (e.g., median, mode), organized by [12].

The incremental aggregates we support are listed as follows: mean, standard deviation, sum, min, max, count, distinct count, trend, time since last time, last value, skew, and feature generation derived from previous aggregates.

Considering continuous tuples from 4G networks are deemed as inputs of downstream of the streaming system like Apache Storm [13], the featre serving automatically will trigger an incremental aggregation pipeline to compute the statistics instantly, without recomputing on a large chunk of historical raw data from scratch. These results will be stored in a fast key-value database, such as Redis, to offer a prompt fetch for the following predictive service.

Two-Phases FE processing algorithm: We create a *Two-Phases FE processing algorithm*. In phase A, the pipeline precomputes all the incremental intermediates for the required measures; in phase B, the pipeline doesn't do any incremental calculation and directly fetches the intermediates produced in phase A and make a simple calculation to derive the final results. The motivation of separating the two phases is to achieve the real-time outcome of the features and minimize the computation and I/O transmission. The phase A takes place right after a new tuple arrives at a window and propagates the update of the tuple with the previous aggregated value. The phase B relies on a few intermediates, which means it is in the blink of an eye. The prerequisite of this approach is that we know in advance about the sort of aggregates and features that are needed in an application. The customization characteristic of the framework is adopted to improve the performance of the response time. For example, $average$ can be treated as $\frac{sum}{count}$. sum and $count$ are incrementally calculated in the phase A ahead, and The phase B only merges the incremental intermediate and the value of the current tuple, and does a division of sum and $count$, and similarly standard deviation relies on *sum of squares, sum, count*.

Rule-based Real-time Distributed Computing The starting point of a model training&test and model prediction service is the generation of a *user directed acyclic graph* (DAG, also known as workflow), composed of nodes produced by our user in the Web UI. The DAG will be automatically mapped into two separate versions, an offline DAG and an real-time DAG. Offline DAGs are different from real-time DAGs because real-time DAGs do not include all the nodes in offline DAGs, such as model training or test, model evaluation, etc. In a DAG, a node represents a task, which can be mainly classified into two categories, one is feature related, another is model related; and an edge stands for the data dependencies between tasks, which makes the nodes connected in a pipelined manner. Through a few clicks of drag-and-drop and parameter

configuration, a user of ML scientist can model a request in a DAG like Fig. 1.

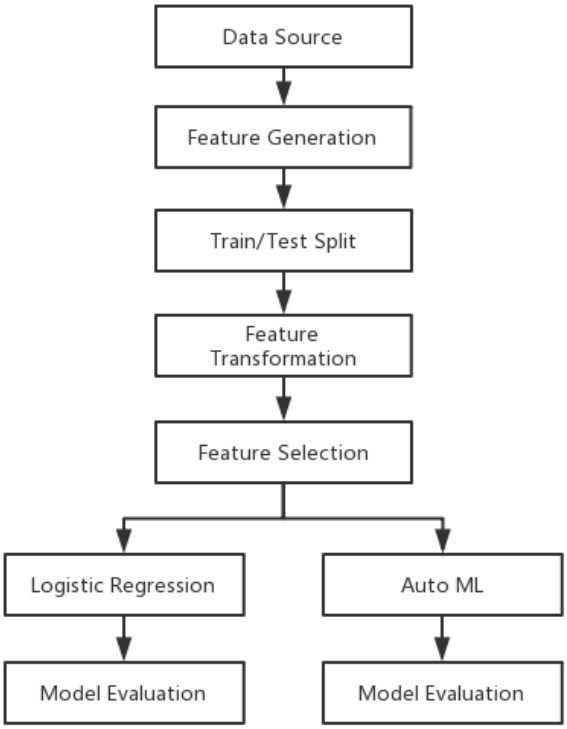


Fig. 1: DAG example

It is worth noting that multiple DAGs could be executed because multiple users could be using the system simultaneously, and a user may create any number of different DAGs at any time, and after model training and test, a user can deploy a DAG service. All DAGs can be executed parallel simultaneously. For offline DAGs, there are many mature resource management and job scheduling systems could help schedule jobs of offline DAGs, such as Hadoop Yarn [14], Apache Mesos [15]. For real-time DAGs, a distributed computing framework is required for adding and removing real-time DAGs dynamically.

The benefit of the rule-based approach is that users only need to draw a user DAG and configure parameters, no need of writing codes. The details will be discussed in Chapter 3.

B. Incremental Windowing

Windows are at the core of processing streams [8], [16], in this paper, we employ an *incremental update* mechanism that can embody incremental changes of the newly arrived tuple and update the intermediates of the measure attributes.

Among the three types of sliding windows: Hopping, Tumbling, and Overlapping, we focus on *Tumbling windows* (also called Fixed windows). Tumbling windows are defined by a static window size, and the window size equals the sliding period [17], [18]. The tumbling window processes queries in a non-overlapping manner. The only difference between our tumbling window and traditional tumbling window is that traditional tumbling window waits until window time is over, and process only once when all the tuples belong to the window is collected. We create an incremental tubmling window, once an incoming tuple arrives, it will get processed at once, so it does not cover the entire input when a windows starts. The current window tumbles on a window-size basis. when the last window time is over, and the window is empty. As time goes by, the input tuples arrives, the window gets filled in real time until it is full. Therefore, we achieve better real time performance by not waiting for the whole length of a window size. For example, a one-minute tumbling window incrementally processes tuples that occurred in one minute and slides after the minute passed.

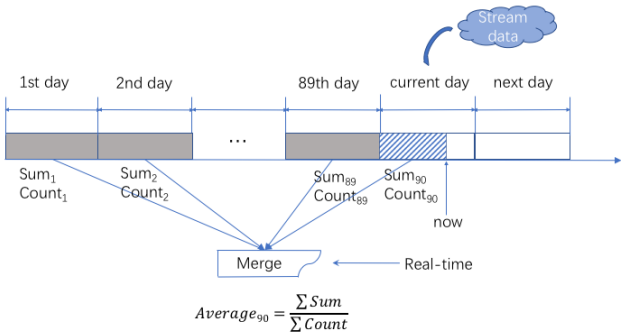


Fig. 2: Windowing design and merge, with an example of computing the average of 90 days.

Windowing chops up an unbounded stream into finite dataset based on time or tuples, and then process each window as a separate group [19]. We support four types of windows sizes, one day (daily), one hour (hourly), one minute, and one second. The upper limits of caching number for each window are 180 days, 24 hours, 60 minutes, and 60 seconds, and we require features within the longest time span of 180 days be available in 50 ms at most. These various window sizes are fairly enough for generating features for possible user queries we collected.

The tumbling window offers an advantage of simplicity for later window merging that joins integer number of windows. The window merging logic subjects to both the window partitioning and sliding methods, it regards the current day as an integer day no matter whether it has finished, then glues the remanent windows together. If a query is about to fetch a feature in the past 90 days, a combination output of eighty-nine-days windows and one incomplete accumulating window will be returned as shown in Fig. 2.

IV. Rule-based Real-time Distributed Computing

It is hard to let users do custom coding and implement the model training and serving especially when requirements change frequently. With the combination of real-time technologies of *resource scheduling, rule engine, RPC HA service*,

the rule-based real-time distributed computing framework is established.

Resource scheduling involves allocating system resources according to the computational load of given tasks, which is a crucial technique for providing high performance distributed computing [20], [21]. Scheduling of tasks in distributed computing systems aims to schedule when and where each task should be executed in a cluster. As can be seen in Fig. 3, the feature service master does the scheduling by checking which worker has the least running jobs.

Rule engines [22] are a mechanism for executing "business rules", interprets complicated condition/action statements, often in the form of "if/then" notations. Rules are stored in a rule database, then the engine watches over the input stream, matches the data against the existing rules in real time, and then execute appropriate actions when conditions are met and therefore can keep data moving at low latency [22]. The biggest advantage of rule engines is that they decouple business logic and application code, which means business logic doesn't need to be hard-coded into the program with procedural languages. All the business logic is centralized, so it is easier to maintain, enhance and update new logic, which is a pluggable component that doesn't have to restart the system for rule changes or deploying new executable rule.

In rule engines, we make one rule represent one DAG executor or one Node executor. A worker and DAGs are of one-to-many relationship. Assume a rule is a DAG, when building a rule engine, and deploying a new DAG to a worker, to decide on which worker the DAG should be deployed, the resource scheduler of the master judges on-the-fly which worker is running the least number of DAGs, then allocates the DAG to the worker. So the association between a DAG and a worker is a one-to-one mapping.

RPC is a protocol that allows one program to request a service of a program located in another computer on a network as if it were a normal local program. We use a cross-language RPC framework as a fundamental technique for distributed services. It provides high availability (HA) in the production environment and supports load-balance. We develop the distributed framework by designing a master and many workers on the cluster. Each worker or master is redundantly distributed in the cluster , if any worker or master failed, the system still provides normal service. The number of workers on a RPC service lies in the CPU and main memory capacity of it. The logical structure of rule engines and distributed services is illustrated in Fig. 3. A job can be a feature serving node or a real-time DAG executor, the significant characteristic of the real-time distributed computing architecture is that both of feature serving and DAG executors are working on the mechanism of it.

When a message arrives, each message has a "topic" attribute, representing the data source of DAGs, and many DAGs may have the same topic (data source), so the master looks up the topic and gets which DAG the message belongs to. Since a DAG and a worker are of one-to-one relation, the master matches the topic with a worker, then dispatches the message

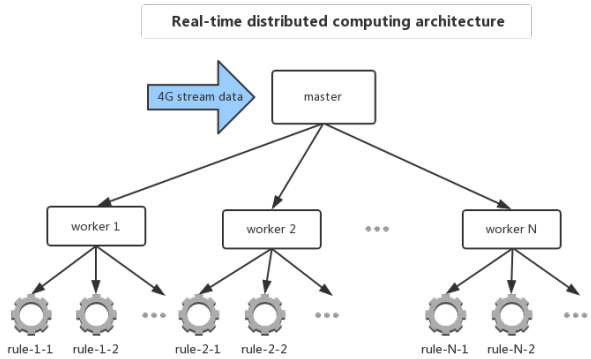


Fig. 3: Real-time distributed computing architecture. If a message from feature serving or a real-time DAG is matched with a rule, a job will be started as a rule executor.

to all the workers with related DAGs, and execute the DAGs concurrently.

Fig. 4 shows how users of Diting interact with the system and work in order. From the user perspective, he or she by drag-and-drop can create an application model as a user DAG, After one click of the "Save" button, the offline system will generate an offline DAG transparently in the background. Then the user click "Train" button, the offline system will run the offline DAG which executes feature engineering and model training and test altogether.

Now, users can click "Deploy Prediction Service" button to add related rules to the real-time service, it will convert the user DAG to a real-time DAG; finally, the user click "Start Prediction Service" button, the system will add the real-time DAG rule to rule-based real-time distributed computing service, and the rule of the service takes effect . The user is free of coding throughout the process.

V. EXPERIMENTAL EVALUATION

We built Diting system on a cluster of 10 servers, each with hardware as follows: 2 x Intel ® Xeon ® E5-2650 v4 of CPU, 16 x DDR4 2133Hz 16GB of RAM, 8 x SAS 2.5" 15K 6Gb 600GB (RAID 10) of storage, 2 x Intel X520 10Gb of network.

We present an experiment of a login model to show the performance of Diting. Login model is an approach to capture outliers of login behaviors in JD over 220 million active users, it captures the sequence of login actions, uses the model of LightGBM to predict the probability of a user account being hacked and at risk of capital losses. The LightGBM is a gradient boosting framework that uses tree based learning algorithms [23].

We evaluate the performance of Diting by doing feature computing for the login model on the real production streaming data, the features calculated by Diting is mainly about the count of login behaviors in the past 2/5/24 hours, and average time of a login action, the distinct count of located cities

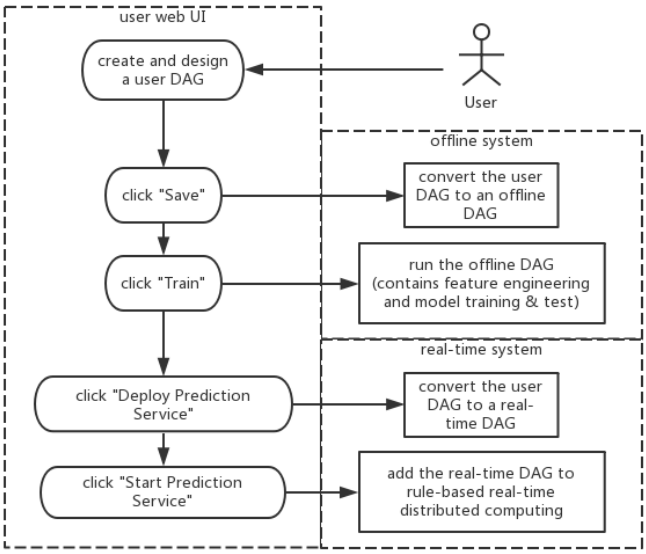


Fig. 4: Diting system flowchart. Users enjoy the real-time end-to-end feature serving and prediction service.

of a login action, etc. There are about 27 features produced in a feature engineering. Fig. 5 shows the prediction service latency of a tuple is done on a dataset of two months, the average time of a feature computing is circa 4.1 ms, which fills the lacunae between offline and real-time feature engineering, and tremendously improves productivity. Traditional methods of developing a pipeline of feature serving take about two months. 100% accuracy is guaranteed as long as the offline and real-time processing are working on the same data.

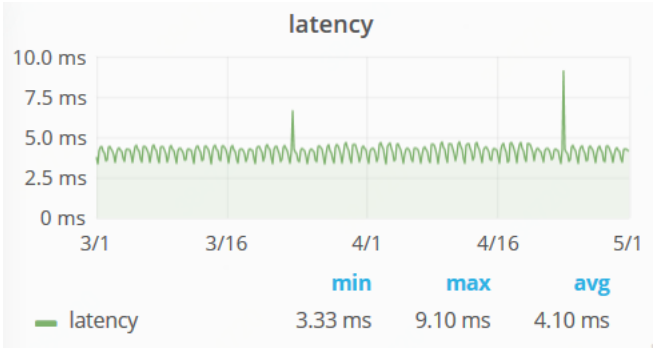


Fig. 5: Average latency running on a real production streaming data source for two months. The two pulses of the figure are due to network instability.

VI. CONCLUSIONS

We presented Diting, a real-time feature serving system with an easy-to-use interface, and a rule-based distributed computing framework. We showed that Diting can create millions of features both in real-time and offline circumstances. We also validated that Diting is real-time in a production cluster with

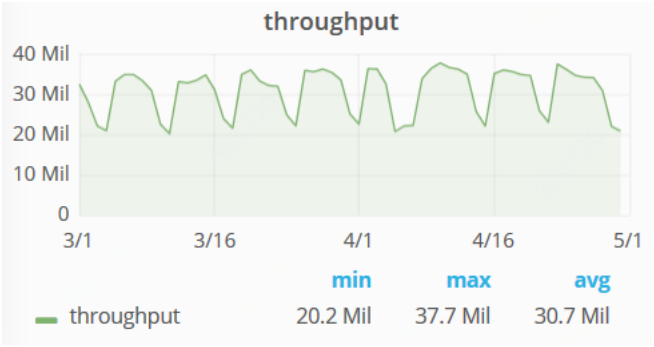


Fig. 6: Throughput aggregated by day running on a streaming data source for two months.

inspiring results. Incremental aggregates of more basic features and more accurate feature computing are our future work.

REFERENCES

[1] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen *et al.*, "Mllib: Machine learning in apache spark," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1235–1241, 2016.

[2] P. Yang, S. Thiagarajan, and J. Lin, "Robust, scalable, real-time event time series aggregation at twitter," in *Proceedings of the 2018 International Conference on Management of Data*. ACM, 2018, pp. 595–599.

[3] Y.-J. Chiang and R. Tamassia, "Dynamic algorithms in computational geometry," *Proceedings of the IEEE*, vol. 80, no. 9, pp. 1412–1434, 1992.

[4] D. Alberts, *Implementation of the dynamic connectivity algorithm by Monika Rauch Henzinger and Valerie King*. Citeseer, 1995.

[5] B. M. Kapron, V. King, and B. Mountjoy, "Dynamic graph connectivity in polylogarithmic worst case time," in *Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 2013, pp. 1131–1142.

[6] P. Bhatotia, U. A. Acar, F. P. Junqueira, and R. Rodrigues, "Slider: incremental sliding window analytics," in *Proceedings of the 15th International Middleware Conference*. ACM, 2014, pp. 61–72.

[7] M. Zaharia, T. Das, H. Li, S. Shenker, and I. Stoica, "Discretized streams: An efficient and fault-tolerant model for stream processing on large clusters." *HotCloud*, vol. 12, pp. 10–10, 2012.

[8] D. J. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik, "Aurora: a new model and architecture for data stream management," *Vldb Journal*, vol. 12, no. 2, pp. 120–139, 2003.

[9] D. J. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. Maskey, A. Rasin, E. Ryvkina *et al.*, "The design of the borealis stream processing engine." in *Cidr*, vol. 5, no. 2005, 2005, pp. 277–289.

[10] T. Akidau, A. Balikov, K. Bekiroğlu, S. Chernyak, J. Haberman, R. Lax, S. McVeety, D. Mills, P. Nordstrom, and S. Whittle, "Millwheel: fault-tolerant stream processing at internet scale," *Proceedings of the VLDB Endowment*, vol. 6, no. 11, pp. 1033–1044, 2013.

[11] A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, S. Kulkarni, J. Jackson, K. Gade, M. Fu, J. Donham *et al.*, "Storm@ twitter," in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. ACM, 2014, pp. 147–156.

[12] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh, "Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals," *Data mining and knowledge discovery*, vol. 1, no. 1, pp. 29–53, 1997.

[13] D. Peng and F. Dabek, "Large-scale incremental processing using distributed transactions and notifications." in *OSDI*, vol. 10, 2010, pp. 1–15.

[14] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth *et al.*, "Apache hadoop yarn: Yet another resource negotiator," in *Proceedings of the 4th annual Symposium on Cloud Computing*. ACM, 2013, p. 5.

[15] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. H. Katz, S. Shenker, and I. Stoica, "Mesos: A platform for fine-grained resource sharing in the data center." in *NSDI*, vol. 11, no. 2011, 2011, pp. 22–22.

[16] P. Carbone, J. Traub, A. Katsifodimos, S. Haridi, and V. Markl, "Cutty: Aggregate sharing for user-defined windows," in *ACM International on Conference on Information and Knowledge Management*, 2016, pp. 1201–1210.

[17] T. Akidau, R. Bradshaw, C. Chambers, S. Chernyak, R. J. Fernández-Moctezuma, R. Lax, S. McVeety, D. Mills, F. Perry, E. Schmidt *et al.*, "The dataflow model: a practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing," *Proceedings of the VLDB Endowment*, vol. 8, no. 12, pp. 1792–1803, 2015.

[18] S. Krishnamurthy, C. Wu, and M. Franklin, "On-the-fly sharing for streamed aggregation," in *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*. ACM, 2006, pp. 623–634.

[19] J. Li, D. Maier, K. Tufte, V. Papadimos, and P. A. Tucker, "Semantics and evaluation techniques for window aggregates in data streams," in *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*. ACM, 2005, pp. 311–322.

[20] M. Kalra and S. Singh, "A review of metaheuristic scheduling techniques in cloud computing," *Egyptian informatics journal*, vol. 16, no. 3, pp. 275–295, 2015.

[21] Z.-H. Zhan, X.-F. Liu, Y.-J. Gong, J. Zhang, H. S.-H. Chung, and Y. Li, "Cloud computing resource scheduling and a survey of its evolutionary approaches," *ACM Computing Surveys (CSUR)*, vol. 47, no. 4, p. 63, 2015.

[22] M. Stonebraker, U. Çetintemel, and S. Zdonik, "The 8 requirements of real-time stream processing," *ACM Sigmod Record*, vol. 34, no. 4, pp. 42–47, 2005.

[23] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," in *Advances in Neural Information Processing Systems*, 2017, pp. 3146–3154.

Achieving consistent real-time latency in a collocated commodity virtual machine environment by LLC partitioning

Chung-Fan Yang
Department of Computer Science
University of Tsukuba
Tsukuba, Ibaraki Japan
sonicyang@softlab.cs.tsukuba.ac.jp

Oscar Garcia
Department of Computer Science
University of Tsukuba
Tsukuba, Ibaraki Japan
oscar@softlab.cs.tsukuba.ac.jp

Yasushi Shinjo
Department of Computer Science
University of Tsukuba
Tsukuba, Ibaraki Japan
yas@cs.tsukuba.ac.jp

***Abstract*—In collocated environments for real-time (RT) and non-RT services, processor cores are often isolated to guarantee real-time requirements. Allocation of the last level cache (LLC) is an emerging problem for consistent latency. This paper shows that consistent real-time latency in a collocated commodity virtual machine environment is achievable by appropriate LLC partitioning and core allocation. In a Linux KVM-based hosted environment, the authors have implemented two LLC partitioning techniques: dedicating cache to critical RT services and allocating a pollute buffer to non-critical non-RT services by using Intel Cache Allocation Technology (CAT). Experimental results show that it was feasible to run both hard and soft real-time services using these cache partitioning techniques when exclusive cores were allocated to real-time virtual machines. It was also feasible to run soft real-time services using these LLC partitioning techniques without allocating exclusive cores.**

***Index Terms*—Real-time, virtual machines, LLC, resource isolation**

I. Introduction

People use real-time (RT) and time sensitive networking services including Voice over IP (VoIP) and interactive Web searching everyday. Such servers require low latency and low variance of latency, and have soft or hard real-time requirements.

It is a common practice to run RT and non-RT services together in shared machines [1], [2]. Because these services have different timing requirements and complexity, we sometimes have to run multiple operating systems (OSs) using a hypervisor. For example, Jailhouse [3], [4] is a hypervisor that is designed to run RT applications on IA64 and ARM processors. It partitions CPU cores and memory resources of a Linux system statically. The main partition executes Linux and others execute additional RT OSs or bare-metal RT programs in parallel for achieving real-time requirements. Linux KVM and Xen are also used for different timing requirements [5]–[10]. In a data center, using a hypervisor is a must for effective management.

In a collocated environment of RT and non-RT services, many approaches focus on the computational resource and allocate exclusive CPU cores to the RT threads of an RT service. Some of them are designed for virtual machine (VM) environments [5], [7]–[10]. These approaches for virtual machine environments, however, do not take into account the last level cache (LLC). Other approaches focus on the LLC and allocate exclusive LLC partitions to an RT service [11]–[13]. However, these approaches of LLC allocation do not consider virtual machine environments.

In this paper, we describe an approach to a consistent real-time latency in a Linux KVM-based hosted environment. We focus on LLC allocation by making maximum use of advanced hardware support, Intel Cache Allocation Technology (CAT) [14]. This hardware enables partitioning LLC and allocating some LLC partitions to a group of threads. We control this hardware through resctrl, a resource control framework and interface of Linux [15].

The contribution of the paper is an experimental demonstration showing that consistent real-time latency is achievable using the CAT in a Linux KVM-based hosted environment. This paper shows the effect of the CAT in the hosted virtual machine environment that ran a critical real-time networking service and non-critical disturbing tasks together. We compare and evaluate two LLC partitioning techniques in conjunction with two CPU core allocation methods.

II. Implementing LLC partitioning with Intel CAT

The cache pollution problem has been evaluated and confirmed harmful for either throughput or latency sensitive workloads on both uni-processor and SMP platforms [11]–[13]. In our particular scenario, we run a critical RT service and other non-critical non-RT services together in a system and evaluate two LLC partitioning techniques that solves the cache pollution problem.

A. Software controlled cache partitioning

We implement LLC partitioning techniques in an Xeon processor, which has an ability to partition the LLC, called Cache Allocation Technology (CAT) [14], [15]. A CAT-enabled processor has a number of pre-defined LLC partitions and classes of services (COSs). Each partition can be associated with one

or multiple COSs. Each thread has the attribute of a COS and a CPU core has the register of a COS. When a kernel executes a thread, the kernel sets the COS attribute of the thread to the COS register of the CPU core. The core executes the thread by accessing the LLC partitions that are associated with the COS. The kernel accesses the machine specific registers (MSRs) to manage and organize the mapping of LLC partitions, COSs and processor cores.

Linux kernel version 4.10 and later provide resctrl, a framework and interface of resource control. This allows system administrators to change the mapping of LLC partitions, COSs and processor cores. In addition, this interface provides a mechanism to monitor the LLC usage for each task, and identify cache pollution. In this paper, we control the LLC partitioning based on the Intel CAT through this framework.

B. Cache dedication and pollute buffer allocation

A typical approach to solve the cache pollution problem is cache partitioning. We reimplement the following two LLC partitioning techniques:

- Dedicating cache to critical RT services [11], [13].
- Allocating a pollute buffer to non-critical non-RT services for containment [12].

The former technique guarantees a required size of cache to critical RT services and eliminates cache pollution. However, it is difficult to find the best cache size while maintain both low latency for RT services and high throughput for non-RT services. It is easier to implement the latter technique than the former one because no estimation of cache size is needed. We can fix the size of a pollute buffer without analyzing RT services.

Figures 1 and 2 show the implementations of these two cache partitioning techniques in a hosted virtual environment using the Intel CAT. In both figures, we run a critical RT service in an RT VM and two non-critical non-RT services in two non-RT VMs. Each VM has two virtual CPU (vCPU), which correspond to two host threads. In addition to vCPU threads, the host OS has house-keeping threads that perform miscellaneous tasks of the host OS. For example, the house-keeping threads execute the timer subsystem tasks and RCU callbacks.

In Figures 1 and 2, we use two COSs, COS 1 and COS 2, for cache partitioning. Figure 1 shows dedication of cache to the critical RT service. The LLC partitions of COS 2 are dedicated to the threads of the RT VM, and the LLC partitions of COS 1 are for other threads. Figure 2 shows allocation of a pollute buffer to the non-critical non-RT services. The LLC partition of COS 2 is allocated to the threads of the non-RT VMs, and the LLC partitions of COS 1 are for other threads.

C. CPU core allocation methods

Besides two LLC partitioning techniques, we examine the effect of CPU core allocation methods. Concretely, we use following two production methods in Linux:

- The RT prioritized method. This method uses the PREEMPT_RT patch [16] and executes the threads of RT services with real-time priorities. When a kernel performs a context switch, the kernel replaces the COS register of a core.
- The exclusive core method [3], [8]. In addition to the PREEMPT_RT patch, this method allocates an exclusive core to a group of host RT threads. The COS register of a core keeps a fixed COS value.

The later method produces better real-time performance than the former method. However, the later method has disadvantages: low utilization of exclusive RT cores and low throughput of collocated non-RT services [17].

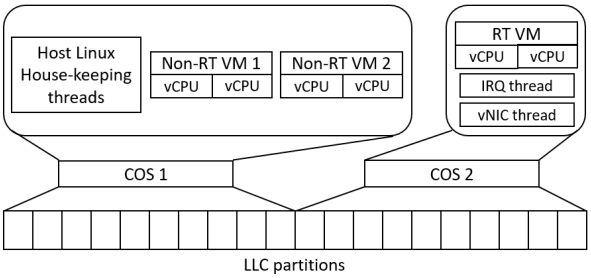


Fig. 1. Dedicating cache to RT service.

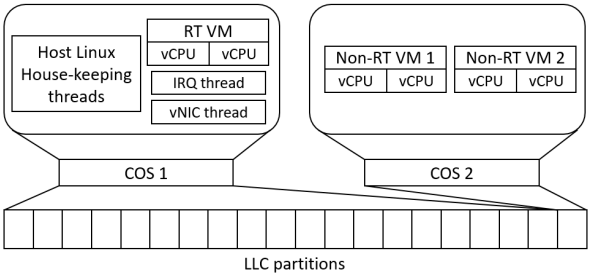


Fig. 2. Allocating a pollute buffer to non-RT services.

III. Experiment and evaluation

A. Experimental setup

We have performed experiments using a simple RT server in the experimental environment shown in Figure 3. We ran netperf [18] as the critical RT service and measured mainly the latency of the server. We used stress-ng [19] as a non-critical non-RT service, which caused noisy cache accesses. We gave stress-ng the following parameters:

```
stress-ng --cache 4 --vm 1
```

With the option "`--cache 4`", the stress-ng program spawned four threads that performed random wide spread memory read to thrash the CPU cache. With the option "`--vm 1`", the stress-ng program spawned a thread that continuously called mmap() and munmap() system calls and wrote to the allocated memory.

We ran a server of netperf in an RT VM and a single instance of stress-ng in a non-RT VM. We ran two instances of non-RT VMs because using two VMs yielded the maximum

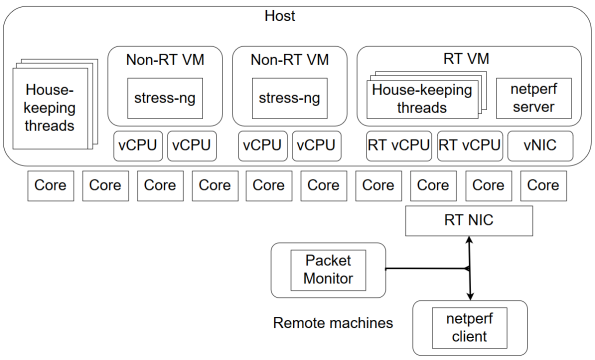


Fig. 3. Experiment environment.

TABLE I
POLICY AND RT PRIORITIES OF THREADS.

Thread	Policy	RT Priority
IRQ thread of RT NIC	SCHED_FIFO	95
RT VM vNIC thread	SCHED_FIFO	94
RT VM vCPU threads	SCHED_FIFO	90
Others	SCHED_NORMAL	–

throughput in terms of the number of instructions per cycle (IPC). Each VM had two vCPU threads. The RT VM had a virtual network interface card (vNIC) thread. The host OS had house-keeping threads as described in Section II-B. The priorities of these threads were shown in Table I.

In the experiments, we used two physical machines. The host machine is specified in Table II. The client machine has an Intel i7-3820 processor with 32GB of RAM. The host OS of virtual machines and the OS of the client machine were Linux version 4.14.34 with the PREEMPT_RT patch. The guest OSs of virtual machines were also the same version of Linux. In the VM host, we disabled hyperthreading, c-states, and p-states to eliminate fluctuations in the hardware [20]. We used the same parameters of Linux and KVM in the reference [6], including enabling full tickless kernel and utilizing hugepage for virtual machine memory regions.

As shown in Figure 3, two PCs were connected with a network, which consisted of 10GBASE-LR Ethernets over optical fiber. We used Intel X520 Ethernet converged network adapters as the NICs. The maximum transfer unit (MTU) of the network was set to the default value, 1500 bytes.

B. Metrics

In the experiments, we measured the following items.

- The maximum latency of the critical RT service. We use this to evaluate the hard real-time performance.
- The 99th percentile latency of the critical RT service. We use this to evaluate the soft real-time performance.
- The LLC miss rate of the real-time virtual machine.
- The IPC of non-RT virtual machines. We use this to evaluate the throughputs of non-RT virtual machines.

We measured the latency of the netperf server with a hardware monitor (an Endace DAG 10X2-S card [21]). We chose to use the hardware monitor because it had no probe effect. The network in Figure 3 consisted of two optical links. Each link had an optical splitter that divides signals into two destinations. One destination was a network peer and the other destination was the hardware monitor. The hardware monitor took both the request and the reply packets, timestamped them at a resolution of 4 ns, and saved them into a file. Note that the obtained results included delays in the NIC of the server, but did not include any delays on the client side.

TABLE II
SPECIFICATION OF VM HOST.

Component	**Specification**
CPU	Intel Xeon 2630 v4 @ 2.2Ghz 10C/10T
Memory	32GB DDR4
LLC Size	20MB
Number of LLC partitions	20
Number of COS	16
NIC	Intel X520 10 Gigabits NIC

We measured the LLC miss rate and IPC with the perf command [22], [23], which took values from hardware performance counters. We have chosen IPC as a throughput metric rather than CPU utilization because IPC reflected effective throughput better in our preliminary experiment. In our experimental environment, we obtained the maximum IPC using two non-RT VMs. When we increased the number of non-RT VMs to three and four, the CPU utilization became larger but the IPC became smaller.

In a single test, the netperf client sent 5000 request messages to the netperf server, and the netperf server sent 5000 reply messages to the netperf client using TCP. The size of a single message was 64 bytes. We have slightly modified the client of netperf, which sent requests at variable inter-arrival times. In our experiments we tested from 1 to 10 millisecond. If the cache pollution program stresses at a fixed rate, its impact to the LLC is unchanged. Such a fixed impact can lead to steady results. We should avoid this (by varying intervals) because we measure the latency variance using multiple LLC partitioning techniques. When the client sent request messages at a shorter inter-arrival time, the LLC retained more contents of the RT service. When the client sent messages at a longer inter-arrival time, the LLC retained fewer contents of the RT service.

C. LLC partitioning techniques and core allocation methods

We compared performance using the following LLC partitioning techniques:

- No partitioning. The entire LLC was shared by all VMs and the host OS.
- Dedication of cache to critical RT services as shown in Figure 1. The LLC was divided into two partitions. The first half (10 MB) of LLC was allocated the RT VM. The other half of LLC was shared by the other VMs and the host OS.
- Allocation of a pollute buffer as shown in Figure 2. The LLC was divided into two partitions. The first 1 MB of LLC was allocated to the non-RT VMs as a pollute buffer

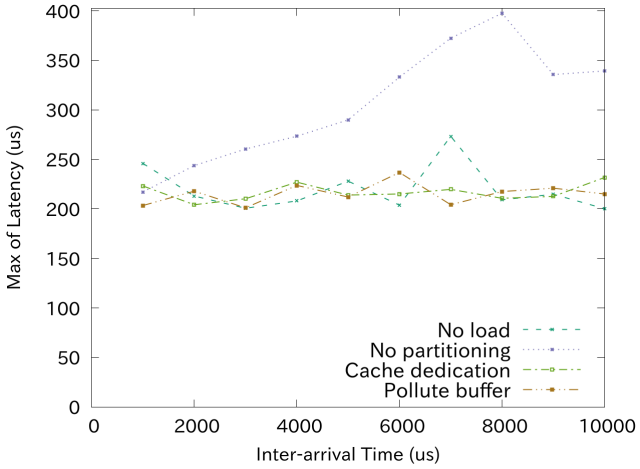


Fig. 4. Maximum latency with exclusive core method.

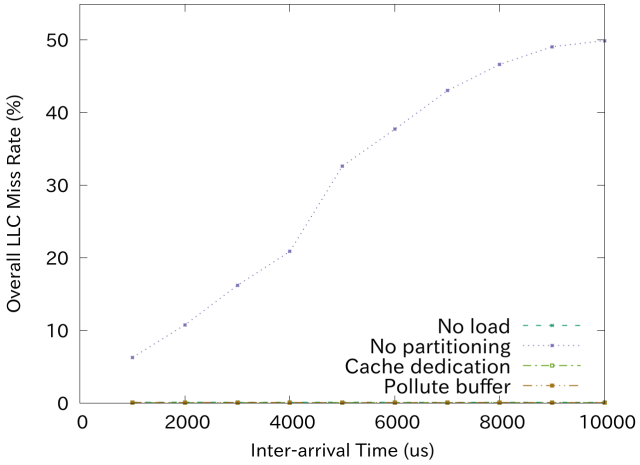


Fig. 6. LLC miss rate with exclusive core method.

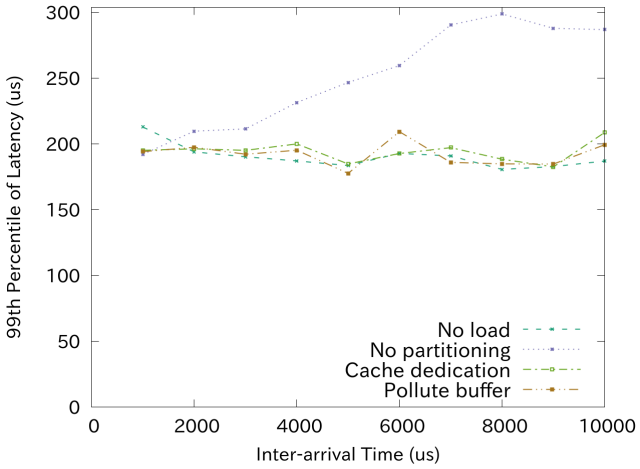


Fig. 5. 99th percentile latency with exclusive core method.

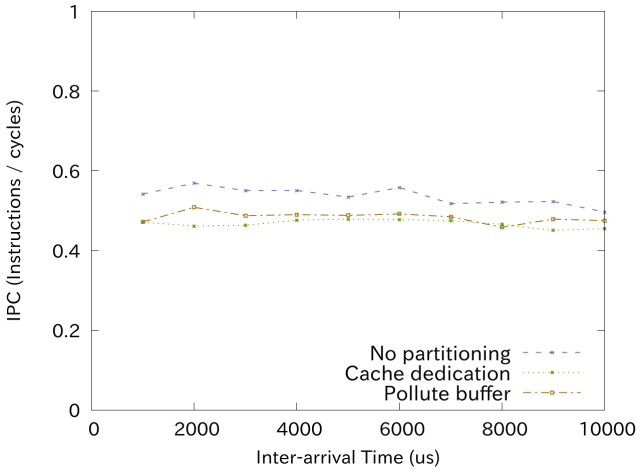


Fig. 7. IPC of non-RT services with exclusive core method.

as described in [12]. The rest of LLC was shared by the RT VM and the host OS.

We also measured performance without running the cache pollution programs (no load).

In conjunction with these LLC partitioning techniques, we tried the following CPU core allocation methods as described in Section II-C:

- The exclusive core method.
- The RT prioritized method.

D. Results using the exclusive core method

In this subsection, we compare the LLC partitioning techniques using the exclusive core method as a core allocation method. In this core allocation method, we pinned the threads of the RT VM, the IRQ and the IRQ threads of the NIC to two RT cores. We pinned the threads of each non-RT VM to two cores. We also pinned house-keeping threads including RCU callbacks to the other cores.

Figures 4 and 5 show the maximum and 99th percentile latency of the netperf server, respectively. Under no load condition, the latency was not affected by the inter-arrival time of requests. Under load condition, the latency was affected by the inter-arrival time of requests with no LLC partitioning technique. By using the two LLC partitioning techniques, we obtained consistent latency. Because both the maximum and 99th percentile latency were consistent, we conclude that it was feasible to run both hard and soft real-time services with these LLC partitioning techniques.

Figure 6 shows the LLC miss rate of the real-time virtual machine. Using no LLC partitioning technique, the LLC miss rate increased up to 50% as the inter-arival time became longer. Using the two LLC partitioning techniques, the LLC miss rate became zero. We did not see any differences between these two LLC partitioning techniques in these experiments.

Figure 7 shows the total IPC of non-RT services. They were around 0.5 and did not vary as the inter-arival time changed.

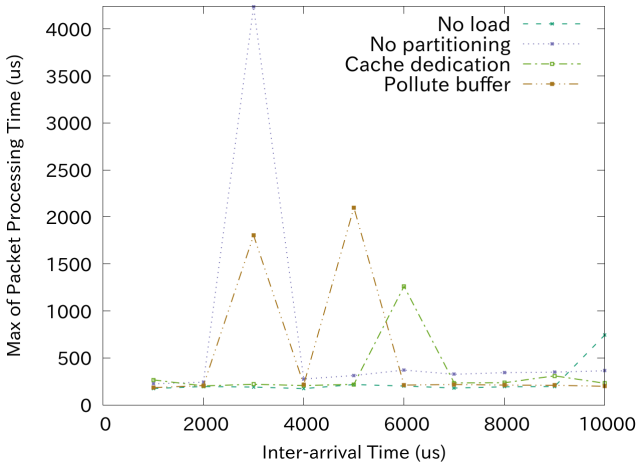


Fig. 8. Maximum latency with RT prioritized method.

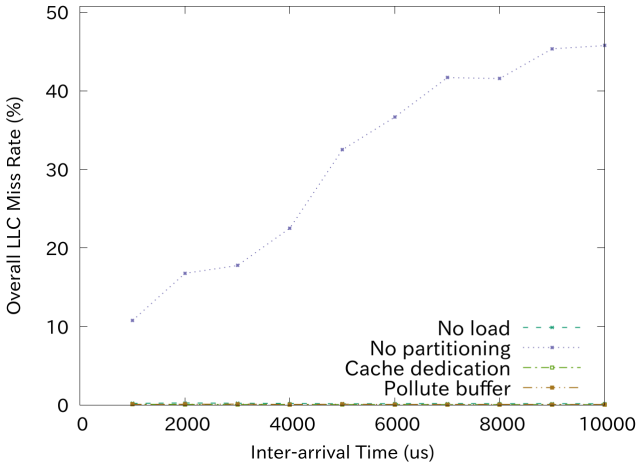


Fig. 10. LLC miss rate with RT prioritized method.

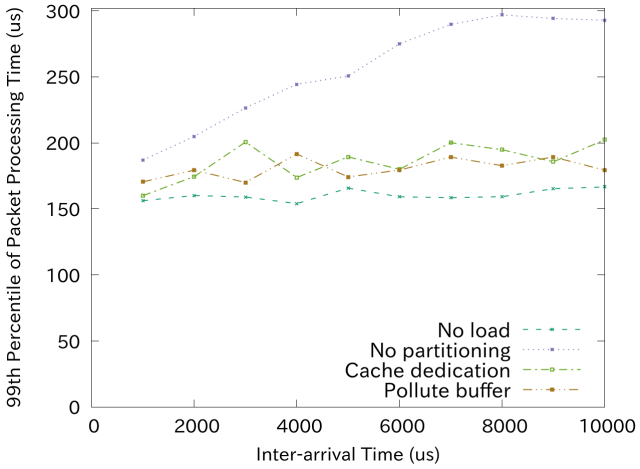


Fig. 9. 99th percentile latency with RT prioritized method.

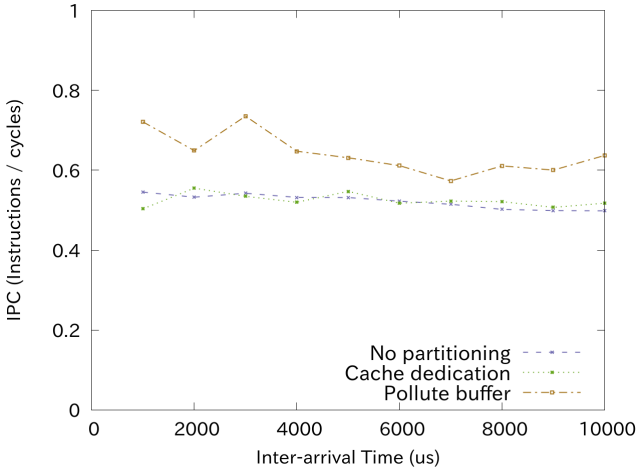


Fig. 11. IPC of non-RT services with RT prioritized method.

E. Results using the RT prioritized method

In this subsection, we compare the LLC partitioning techniques using the RT prioritized method as a core allocation method. In this core allocation method, we did not pinned the threads of RT VMs, the IRQ and the IRQ threads of the NIC to cores. Any core was able to run any thread.

Figure 8 shows the maximum latency of the netperf server. Under no load condition, we obtained a spike when the inter-arrival time was 10 milliseconds. Under load condition, unlike in Figure 8, we obtained spikes up to 4 milliseconds using any LLC partitioning techniques. We conclude that it was not feasible to run hard real-time services with these LLC partitioning techniques using the RT prioritized method as a core allocation method.

Figure 9 shows the 99th percentile latency of the netperf server. This is very similar to Figure 5. We conclude that it was feasible to run soft real-time services with these LLC partitioning techniques using the RT prioritized method as a core allocation method.

Figure 10 shows the LLC miss rate and is also similar to Figure 6. Figure 11 shows the IPC of non-RT services using RT prioritized method. They were around 0.5 to 0.7 and did not vary as the inter-arival time changed. They were slightly higher than those in Figure 7 using the exclusive core method.

IV. Related work

Early systems use page coloring techniques to allocate the cache of the processors that have uniform set-associative cache architecture. The paper [12] proposes a software-based LLC partitioning technique using a pollute buffer and evaluates the technique on a single core processor. This paper uses a page coloring technique at the sub-process level. The paper [11] uses a page coloring technique to allocate the cache for HPC applications.

These techniques do not work on the modern processors that have non-uniform cache architecture (NUCA). Processor vendors do not publish the internal mechanisms of NUCA and it is obscure and difficult to peform reverse-engineering of NUCA [10]. In this paper, we do not use a page coloring technique but use software configurable LLC partitioning mechanism as described in Section II.

The paper [13] uses multiple resource allocation techniques, including cache partitioning techniques, for Web services in Linux. LLC partitioning is dynamically controlled to keep up with service level objectives (SLOs). This paper does not consider virtual machine environments. In contrast, in this paper we have implemented and evaluated LLC partitioning techniques in a hosted virtual machine environment.

The paper [24] shows methods to efficiently schedule resources in network function virtualization (NFV) on a DPDK based system. The proposed scheduler is able to reduce the effect of noisy neighbors and cache contentions of the system to comply given SLOs in NFVs. In this paper, we tackle a similar problem, while targeting a real-time hosted virtual machine environment.

V. Conclusion

In this paper, we showed that consistent real-time latency in a collocated commodity virtual machine environment is achievable by appropriate LLC partitioning and core allocation. In a Linux KVM-based hosted environment, we have implemented two cache partitioning techniques: dedicating cache to critical RT services and allocating a pollute buffer to non-critical non-RT services by using Intel Cache Allocation Technology (CAT).

In the experiments, we measured the maximum and 99th percentile latency of a simple network service. Experimental results show that it was feasible to run both hard and soft real-time services using these cache partitioning techniques when exclusive cores were allocated to real-time virtual machines. Experimental results also show that it was feasible to run soft real-time services but we obtained spikes using these LLC partitioning techniques without allocating exclusive cores.

We are currently finding the causes of these spikes. In the future, we would like to evaluate these LLC partitioning techniques using time sensitive network applications, such as VoIP servers.

Acknowledgment

This work was partially supported by JSPS KAKENHI Grant Number 25540022 and 16K12410.

References

[1] M. S. Mollison, J. P. Erickson, J. H. Anderson, S. K. Baruah, and J. A. Scoredos, "Mixed-criticality real-time scheduling for multicore systems," in *Proc. of the 10th IEEE International Conference on Computer and Information Technology*, 2010, pp. 1864–1871.

[2] J. H. Anderson, S. K. Baruah, and B. B. Brandenburg, "Multicore operating-system support for mixed criticality," in *Proc. of the Workshop on Mixed Criticality: Roadmap to Evolving UAV Certification*, 2009, pp. 1–11.

[3] R. Ramsauer, J. Kiszka, D. Lohmann, and W. Mauerer, "Look mum, no vm exits!(almost)," *arXiv preprint arXiv:1705.06932*, 2017. [Online]. Available: https://arxiv.org/abs/1705.06932

[4] R. Ramsauer, "Building mixed criticality Linux systems with the Jailhouse hypervisor," in *Proc. of the Embedded Linux conference*, 2017. [Online]. Available: http://events17.linuxfoundation.org/sites/events/files/slides/ELC17-Ramsauer-Kiszka.pdf

[5] J. Kiszka and R. Riel, "Two approaches to real-time virtualization - Jailhouse and KVM," in *Proc. of the Linux Foundation Real-Time Summit*, 2016. [Online]. Available: https://wiki.linuxfoundation.org/realtime/events/rt-summit2016/schedule

[6] P. Zhang, "See what happened with real time KVM when building real time cloud," in *Proc. of the Linux Foundation LinuxCon China*, 2017. [Online]. Available: https://www.slideshare.net/LCChina

[7] A. Crespo, I. Ripoll, and M. Masmano, "Partitioned embedded architecture based on hypervisor: The XtratuM approach," in *Proc. of the European Dependable Computing Conference*, 2010, pp. 67–72.

[8] M. Christofferson, "4 ways to improve performance in embedded Linux systems," in *Proc. of the Korea Linux Forum*, 2013. [Online]. Available: https://pdfs.semanticscholar.org/presentation/d7d1/942f800fc8143cca8c7c6428c42b1d20c70c.pdf

[9] S. Xi, M. Xu, C. Lu, L. T. X. Phan, C. Gill, O. Sokolsky, and I. Lee, "Real-time multi-core virtual machine scheduling in Xen," in *Proc. of the International Conference on Embedded Software (EMSOFT)*, 2014, pp. 1–10.

[10] M. Lee, A. S. Krishnakumar, P. Krishnan, N. Singh, and S. Yajnik, "Supporting soft real-time tasks in the Xen hypervisor," in *Proc. of the 6th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE '10)*, 2010, pp. 97–108.

[11] S. Perarnau, M. Tchiboukdjian, and G. Huard, "Controlling cache utilization of HPC applications," in *Proc. of the international conference on Supercomputing*, 2011, pp. 295–304.

[12] L. Soares, D. Tam, and M. Stumm, "Reducing the harmful effects of last-level cache polluters with an OS-level, software-only pollute buffer," in *Proc. of the 41st IEEE/ACM International Symposium on Microarchitecture*, 2008, pp. 258–269.

[13] D. Lo, L. Cheng, R. Govindaraju, P. Ranganathan, and C. Kozyrakis, "Heracles: Improving resource efficiency at scale," in *Proc. of the ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, 2015, pp. 450–462.

[14] K. T. Nguyen. Introduction to cache allocation technology in the Intel Xeon processor E5 v4 family. [Online]. Available: https://software.intel.com/en-us/articles/introduction-to-cache-allocation-technology

[15] F. Yu, T. Luck, and V. Shivappa. User interface for resource allocation in Intel resource director technology. [Online]. Available: https://www.kernel.org/doc/Documentation/x86/intel_rdt_ui.txt

[16] realtime:start [Linux foundation wiki]. [Online]. Available: https://wiki.linuxfoundation.org/realtime/start

[17] O. Garcia, Y. Shinjo, and C. Pu, "Implementation and comparative evaluation of an outsourcing approach to real-time network services in commodity hosted environments," in *Proc. of SCF International Conference on Cloud Computing*, 2018, pp. 189–205.

[18] Netperf. [Online]. Available: https://hewlettpackard.github.io/netperf/

[19] stress-ng: a tool to load and stress a computer system. [Online]. Available: http://kernel.ubuntu.com/cking/stress-ng

[20] *Intel 64 and IA-32 Architectures Software Developers Manual*. Intel, 2018.

[21] *Endace DAG 10X2-S datasheet*, Endace, 2016.

[22] Perf wiki. [Online]. Available: https://perf.wiki.kernel.org/

[23] A. Melo, "The new Linux perf tools," in *Proc. of Linux Kongress*, 2010.

[24] A. Tootoonchian, A. Panda, C. Lan, M. Walls, K. Argyraki, S. Ratnasamy, and S. Shenker, "Resq: Enabling slos in network function virtualization," in *Proc. of the 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, 2018, pp. 283–297.

TREC4CPS 2018

Proceedings of the 1st Workshop on Trustworthy and Real-Time Edge Computing for Cyber-Physical Systems

Edited by
Aron LASZKA, Abhishek DUBEY, and Aniruddha GOKHALE

Organizers

Program Chair

Aron Laszka, University of Houston

Organizing Committee

Abhishek Dubey, Vanderbilt University
Aniruddha Gokhale, Vanderbilt University
Taylor Johnson, Vanderbilt University
Aron Laszka, University of Houston
Martin Lehofer, Siemens, NJ
Keiichi Yasumoto, NAIST, Japan

Program Committee

The organizing committee served as the program committee. Additionally, we used the services of Dr. Shashank Shekhar (shashankshekhar@siemens.com).

Message from the Organizing Committee

It is our pleasure to welcome you to the **1st International Workshop on Trustworthy and Real-time Edge Computing for Cyber-Physical Systems (TREC4CPS)** collocated with the 39th IEEE Real-time Systems Symposium (RTSS) on December 11, 2018 in Nashville, TN, USA.

The increasing proliferation of Internet of Things (IoT) is giving rise to an ever-increasing volume of data that is being generated by the IoT sensors that reside at the edge of the network. Of specific interest to us are IoT applications found in cyber-physical systems where the streamed information must be processed in real-time to make informed decisions for a wide range of societal and environmental applications. For instance, emergency response systems and smart transportation systems are prime examples of multi-domain smart and connected community applications residing at the edge of networks. The term *Tactile Internet* has also been used to refer to these systems. Conventionally, such systems have been implemented using centralized architectures. However, as the scale and penetration of these data driven applications in the communities are growing, the challenges of these architectures become apparent; for example, the lack of scalability, single points of failure, and saturated communication resources. Moreover, the sporadic and uncertain arrival patterns for the IoT data streams complicates real-time stream processing because resources must be provisioned on-demand to fuse multiple temporally-unsynchronized data streams. Second, the temporally sensitive nature of the data, the resource constraints on IoT devices, and the large volumes of generated information make it problematic to always move these information streams to a centralized cloud data center that may be multiple network hops away with fluctuating bandwidths and hence the incurred delays, which is detrimental to the cyber physical systems.

The purpose of the TRC4CPS workshop is to share new ideas, experiences and information about research and development in realizing trustworthy and real-time edge computing systems. The workshop comprises an invited talk by Dr. Aaron Paulos from Raytheon followed by two sessions of three papers each that handle different challenges in this realm. Finally, we close the workshop with a panel discussion on the topic of “Taming Uncertainty in Edge Computing to Assure Performance and Reliability.”

We would like to thank *all* the authors who submitted their work to TREC4CPS, without whom this workshop would not be possible. Next, we would like to thank the PC for their hard work in reviewing the papers. We would also like to thank the RTSS 2018 Workshop Chair Liliana Cucu-Grosjean for her guidance and encouragement. Finally, we would like to thank the RTSS 2018 General Chair Isabelle Puaut, Program Chair Rob Davis, and the remainder of the RTSS 2018 Organizing Committee for their support. We wish you an interesting and exciting workshop and an enjoyable stay in Nashville, TN.

TREC4CPS 2018 Organizing Committee

TREC4CPS 2018 Technical Program

TREC4CPS 2018 Keynote

Aaron Paulos

Supporting Resiliency and Timeliness in Edge Applications with Dispersed Computing

BIO: Mr. **Aaron Paulos** is a principal investigator and computer scientist at BBN Technologies. At BBN, he investigates transparent adaptive computing paradigms as a method to improve the security, resiliency, and trust properties of distributed applications. Over the last decade, Aaron has supported a number of AFRL, DARPA, and IARPA research efforts. Prior to joining BBN, he researched fault tolerance and assistive technologies for the blind at the ECE department of Carnegie Mellon University. Aaron has an M.S. from the Information Network Institute at CMU and a B.S. in Computer Science from the University of Pittsburgh. He is also a member of the ACM and IEEE.

ABSTRACT: In mission-critical applications that rely upon edge components, any dependence on centralized data centers and shared networks can influence the overall efficiency and efficacy of the application. Losses of links, congestion, or jitter during data delivery, and over-extended CPU and memory are examples of faults and stresses that may cumulatively degrade performance. In this talk, we describe a current research effort that is investigating and prototyping transparent middleware-based approaches for utilizing emerging in-network compute resources to migrate and host computation in optimal locations with respect to demand and resource availability. We then suggest how such capabilities can improve the performance and resiliency of IoT applications.

Supporting Resiliency and Timeliness in Edge Applications with Dispersed Computing

Aaron Paulos
Distributed System Group
Raytheon BBN Technologies
Cambridge MA, USA
apaulos@bbn.com

***Abstract*—In this abstract, we introduce a research effort that explores the multi-attribute trade space covering the placememt and location of computation and data, application performance and resiliency, and resource utilization. We then discuss a concept of how applications that consume large volumes of Internet of Things (IoT) edge data and need to produce timely and reliable results can benefit from this model of *dispersed computing*.**

***Index Terms*—Distributed Computing, Middleware, Edge Networks, Internet of Things (IoT), Real-time Processing**

I. Background

Inexpensive networked IoT devices couple powerful sensors with low-power processors. Such devices are often networked to analytic and control software that execute more complex and computationally expensive logic for reasoning about the sensed data. In typical asymmetric deployments, streams of rich sensor data, potentially in large volumes, will flow from the network edge to a centralized data processing point. At this command point sensor data is analyzed, and small control messages are often sent back to the devices to re-task or fine-tune its operation. In another data flow, analytic results may be delivered to human operators under a strict timeliness requirement (e.g., for search and rescue, threat identification, or weather pattern detection application).

In mission-critical and real-time applications with edge components, the dependence on centralized data centers and the end-to-end networking substrate are key influencers of the overall efficiency and efficacy of an application. Losses of links, congestion, or jitter during data delivery, and over-extended CPU and memeory are examples of faults and stresses that may cumulatively degrade performance. In our current work, we are investigating and prototyping transparent middleware-based approaches for utilizing emerging in-network compute resources to host computation in optimal locations with respect to the demand (see Fig. 1) and to minimize the usage of the network backhaul.

In-network compute resources have the potential to supplement the data centers' massive compute power by widely distributing localized compute capacities throughout the network. This emerging trend offers an opportunity to rethink how a mission-aware computing paradigms can address the challenges faced by data-heavy IoT applications. While our work suggests one approach to transparently managing tasks and data, many complementary techniques can help to mitigate quality-of-service (QoS) concerns. Perhaps most closely related is the promise of edge and fog computing [1], [2]. Other mature approaches such as data filtering or aggregation [3] can reduce the total volume of IoT sensor data. Select resiliency capabilities such as forward error correction [4] and QoS techniques [5] can help to maximize the predictability and robustness of operations.

This work is being supported by DARPA under Contract No. HR0011-17-C-0049.

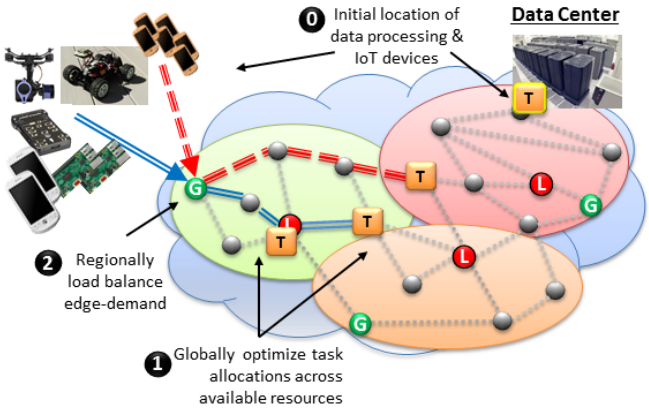


Fig. 1. Concept of Operation: Dispersing IoT Tasks into the Network.

II. A Dispersed Computing Concept for Adaptively Placing IoT Tasks

Let us consider an scenario where a number of sensors are collecting weather and environmental data. Such low-power IoT devices cannot locally perform the analyses to detect when a microburst (weather event) is forming, or when a harmful chemical is released maliciously or leaked accidentally from an industrial source. Therefore, it is common to backhaul a continuous stream of data from the sensors to a data center for processing. In-turn, human operators rely on the data center service to provide the analysis results in a timely manner.

The network between the IoTs and the data center and the human operators is assumed to be dynamic – its capacity and load changes, although not instantaneously but not completely predictably either. Furthermore, link and node failures

may also occur. Under these conditions, reliance on a single centralized data center is unlikely to always provide quality analytic results in a reliable and timely manner. At the same time, we believe it is not always the best strategy to move all computation closer to the IoT sources, which may introduce unwanted delay and outage for the time constrained operators.

We are investigating adaptive methods for fitting many application tasks over time into dynamic networks under application and mission constraints. We suggest a simple model for such a placement here. First, the paths to compute locations from the edge-devices and the available compute power (and from application to operators) must be able to accommodate the load. For realism, resources enabling in-network computing and communication should be assumed to be non-uniform. Second, applications may be composed of more than one interconnected task which may be replicated into multiple locations. Replication will require resource management algorithms to reason about data consistency overheads. Furthermore, interconnected tasks (e.g., a micro-services service chain) with strong levels of *affinity* (e.g., communication depedencies) implies that if tasks are not placed together the collective performance could degrade further.

To effectively disperse computation and data into the network, we are developing decentralized algorithms that balance resource availability and utilization, task or resource affinities, and end-user demand or expected compute and communication load, as an aggregate program [6]. The program peforms continuous monitoring of the resource state and application performance, and produces plans and actions for managing state. Our design is inspired decentralized divide-and-conquer architectures, where mixed-tempo global and a regional algorithmic layers reason about the constraints suggested above.

At the global layer, we are formulating distributed constraint optimization problems (DCOP) that function as a slower paced overall strategic reasoning system. As an output, the global layer provides resource allocation plans indicating how much application computation and data can be placed within smaller logical areas in the nerwork, called regions. At the regional layer, we are applying well studied load balancing techniques to rapidly respond (i.e., tactically manage) to on-going demand while staying within the bounds of the DCOP plan. To support in-network task allocation the regional layer actively manages the lifecycle of local computation. DNS delegation redirects edge-IoT devices' name service lookups to newly started local resources that can process their data. To ensure stability across the global and regional layers, we are applying control theoritic analyses to identify sources of destabilizing volitility.

Early results using our current prototype is promising. We highlight a result of an experiment (java-based simulation) in Fig. 2, which demonstrates application migration to avoid overloaded network paths leading to a centerlized data center. In this experiment, the data center has infinate capacity and there are 100 nodes of varying capacity within the network by design. At the start, the clients use one or more of the three streaming applications hosted in the data center. Each of these streaming applications can be migrated into the network. There is a network region labeled C, from which paths to the data center are severerly congested. This impacts the clients of the three services. As a client scoring function, we consider the network throughput achieved by three types of streaming sensors (i.e., the path between the edge and application). In the case where throughput drops below a desired threshold, we score a failed request. In the case where the sensor throughput is above the threshold, we score a success.

In Fig. 2, we graph the results of a 20 minute execution using a dense test topology and the configuration above. Initially, due to experiment design, some of the client requests fail. As our system measures demand and congestion, the global and regional algorithms satisfice demand and trigger task migtration. Near time 400 seconds, task migration completes and future failures are avoided.

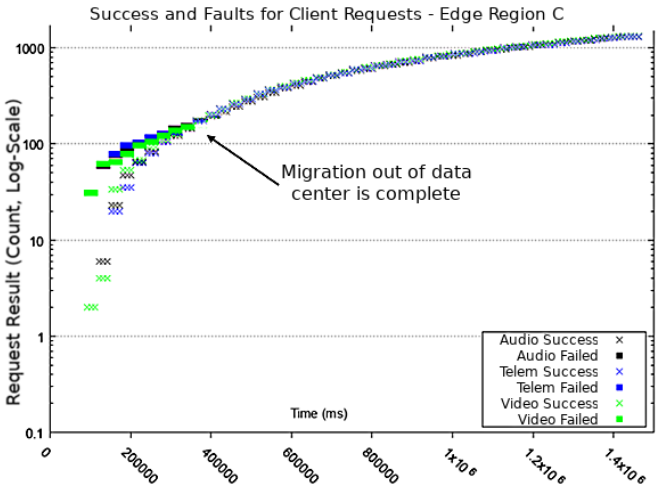


Fig. 2. Experiment Showing Benefits of Task Migration into an Edge Region.

ACKNOWLEDGMENT

This work is being supported by DARPA under Contract No. HR0011-17-C-0049.

REFERENCES

[1] M. Jia and J. Cao and W. Liang., "Optimal Cloudlet Placement and User to Cloudlet Allocation in Wireless Metropolitan Area Networks," IEEE Transactions on Cloud Computing. vol. 5, pp. 725-737, 2017.

[2] C. Mouradian, et. al., "A Comprehensive Survey on Fog Computing: State-of-the-Art and Research Challenges," IEEE Communications Surveys Tutorials. vol. 20, pp. 416-464, 2018.

[3] R. Rajagopalan and P. K. Varshney, "Data-aggregation techniques in sensor networks: A survey," IEEE Communications Surveys Tutorials. vol. 8, pp. 48-63, 2006.

[4] D. Nguyen and T. Tran and T. Nguyen and B. Bose, "Wireless Broadcast Using Network Coding," IEEE Transactions on Vehicular Technology. vol. 58, pp. 914-925, Feb. 2009.

[5] M. Garca-Valls, P. Basanta-Val and I. Estvez-Ayres, "Adaptive real-time video transmission over DDS," IEEE International Conference on Industrial Informatics. Osaka, pp. 130-135, 2010.

[6] D. Pianini and J. Beal and M. Viroli, "Practical Aggregate Programming with Protelis," IEEE International Workshops on Foundations and Applications of Self* Systems. pp. 391-392, 2017.

Scheduling of Smart Factories using Edge Computing and Clouds

Piotr Dziurzanski, Jerry Swan and Leandro Soares Indrusiak

***Abstract*— Reconfiguration-as-a-Service is an emerging trend for dynamic smart factories. This approach exploits cloud-based services to continuously optimise the performance of manufacturing systems. The edge computing paradigm, on the contrary, aims at performing the whole computation at the edge of the network, close to the data sources. In this paper, a trade-off between these two possibilities is analysed. A value-based criterion is proposed for executing optimisation engine either in a cloud or at the edge. Experimental results determine the ranges for both the cloud computation cost and the edge computer's speed in which manufacturing scheduling leads to higher profits.**

I. Introduction

One of the emerging trends related to smart factories is to migrate some computational tasks (e.g. scheduling of manufacturing processes) from remote clouds in order to be closer to devices that are the source and/or target of such computing, i.e. to the *edge* between the IoT's *things* and the network [1]. One of the key predictions discussed in the IDC report [2] was that in the near future almost half of IoT-created data will be stored, processed, analysed and acted upon at or close to the network edge. This migration is expected to be beneficial in terms of response time, reliability, security and cost effectiveness [3]. It may however be argued that whether a certain computation is to be performed in a cloud or at the network edge should be a dynamic decision. That is, it should be based on the predicted gains and costs of both situational alternatives, rather than decided statically without any situational awareness. In Ismail et al [4], Docker containers were proposed to be executed at the edge. The same containers can be executed in a cloud as well (e.g. by using *IBM Cloud Functions*[1]) even in the case of different os/architecture combinations (thanks to the experimental Docker feature named *Docker manifest*). Therefore, the decision on where to execute a certain container can be made dynamically considering the current edge node utilisation or network bandwidth, and also taking into account the urgency of the computation. In most cases, for efficiency and security reasons it may be beneficial to start computation at the edge, since it decreases network traffic and avoids public/shared networks and servers. However, if the computation performed at the edge progresses too slowly, it can be migrated to a cloud. Such approach requires a method to compare the predicted execution time in both edge and cloud. In this paper we follow a method that predicts the benefits of further manufacturing optimisation proposed in ref. [5]. According to that method, each manufacturing order, which requests the production of a particular commodity, is equipped with a *value curve*, that models the value, expressed in the monetary units, yielded by the manufacturing order over time. This value curve can influence the optimisation process as follows: since further search-based schedule optimisation is occupied with the cost of the cloud nodes performing the computation, it has been shown in that paper that it may be beneficial (grounded in terms of overall monetary cost) to prematurely stop the optimisation and apply the best results found so far. In this paper, we propose to extend that model with the possibility of performing the optimisation at the network edge. Using the proposed technique, on obtaining a manufacturing order, an agent decides not only when to finish the optimisation process, but also whether the computation should be performed at the edge or in a cloud, comparing the predicted monetary gains for all these options.

Department of Computer Science, University of York, Deramore Lane, Heslington, YO10 5GH, York, UK
{piotr.dziurzanski,leandro.indrusiak}@york.ac.uk

[1]https://console.bluemix.net/openwhisk

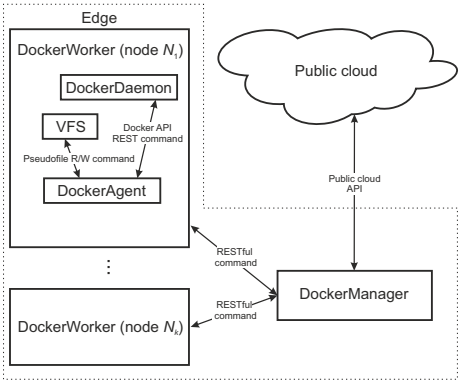


Fig. 1. General scheme of the proposed approach implementation

II. Platform and application model

The class of scheduling problems analysed in this paper concerns manufacturing in which the value gained by an end-user depends on both optimisation solution quality and the time taken by the optimisation process itself. The optimisation process is performed either at the network edge or in a cloud. Suitable application and platform models are proposed below.

A. Platform model

At the network edge, there is a set of k computing nodes $N = \{N_1, \ldots, N_k\}$ capable of executing one or more containers (i.e. each node runs a typical Docker container engine). The nodes are heterogeneous and their response time difference is expressed with so-called *calibration coefficient* ζ_x, $x \in \{1, \ldots, k\}$. ζ_x denotes the ratio between empirically measured response time of a set of container benchmarks on node N_x and the averaged response time of the same set of container benchmarks executed on a reference unit in

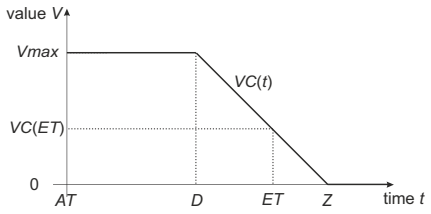


Fig. 2. An example value curve of manufacturing order O

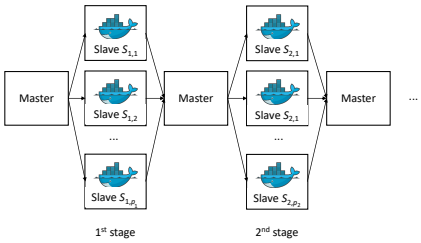


Fig. 3. Stages of the optimisation process

a public cloud serving as an alternative execution platform, as shown in Fig. 1 (the details of this figure are explained later in this paper). The benchmarks' response times on a reference unit include the communication cost and the container initialisation time.

In ref. [5], it was assumed that the schedule optimisation engine was containerised and executed in a public cloud using a function as a service facility, which significantly reduced the initialisation time and monetary cost, in comparison with the more prevalent Containers as a Service paradigm. However, such containers can also be executed at the edge of the network, potentially decreasing the optimisation cost. Only when it is predicted that further local optimisation at the edge is likely to be less beneficial than remote execution, are the containers migrated and executed in the cloud.

B. Application model

The application considered in this paper is related to manufacturing scheduling optimisation in a smart factory. At time instants not known *a priori*, manufacturing orders are submitted. Each of these orders usually concerns the production of several items of a certain product. The role of optimisation is to allocate the manufacturing processes (such as mixing powders, cutting parts etc.) to different machines, select the most appropriate machine modes (e.g. thereby trading production time against energy efficiency) and schedule these processes in time, following the dependency relation between these processes. As discussed for example in ref. [5], such optimisation problems are NP-hard and thus various search-based heuristics are usually applied to find an approximate solution.

Each optimisation process is performed dynamically and concurrently to the manufacturing of the previous orders. Consequently, optimisation results must typically be provided within a limited time span. As long as the factory is busy with the previous orders, the optimisation time does not matter. However, in the case of an idle factory, the time spent on optimisation incurs ongoing factory maintenance costs due to idleness. This phenomenon is well illustrated with the value curve presented in Fig. 2. At time instant AT a manufacturing order is submitted. The maximal possible profit from this order is equal to V_{max}, defined as the excess of revenue over cost and denoted in monetary units. As the factory is busy up to time instant D, processing orders submitted and scheduled earlier, the profit value does not change in interval $[AT, D)$. However, after D, the profit value decreases up to a certain point Z, where it reaches 0. If the optimisation process ends at time ET, the maximal potential profit cannot exceed the value of the curve at ET, namely $VC(ET)$.

III. Optimisation trade-offs

The scheduling optimisation is performed in a master-slave fashion as illustrated in Fig. 3, in sequential stages indexed with $i = 1, 2, \ldots$. At each i-th stage, a set of p_i containers is executed in parallel by slave nodes. The global master coordinates the execution of containers submitted by the users. The master is responsible for serving the incoming requests and allocates the containers to nodes, for example using the algorithms proposed in ref. [6].

All containers $S_{i,y}$, $y \in \{1, \ldots, p_i\}$, are executed either in a cloud or at the network edge. Each container gets the encoded manufacturing order together with the best solutions found so far as its input and after time $t_{i,y}$ returns the minimal value found by the optimisation for the manufacturing cost of that order, $f_{i,y}$, together with the corresponding solution.

If $S_{i,y}$ is executed on edge node N_x, its CPU time slot is proportional to the so-called *CPU shares* $\xi_{i,y} \in \{1, \ldots, 1024\}$ (the value of the maximum share is taken directly from the Docker's *–cpu-shares* flag). Assuming that the sum of all the CPU shares of containers executed on node N_x equals Ξ_x, container $S_{i,y}$ gets $\vartheta_{i,y,x} = \xi_{i,y}/\Xi_x$ of the CPU time of node N_x.

Initially, the execution time of the containers $S_{i,y}$ is difficult to be predict accurately. However, as all these containers are constructed from the same container image and perform optimisation of the same problem size, the workload inside these containers is similar. Thus the response time $t_{i,y}$ of each container $S_{i,y}$ can be measured and used by the master node to predict the future response times at the following stages, as described subsequently.

Due to the change of a potential maximal profit from a certain manufacturing order over time as described by a value curve, a clear trade-off between the optimisation time and the optimisation quality can be identified. As a search-based heuristic keeps the best result found so far and continuously explores the search space up to the fulfilment of a certain stopping criterion, it can provide a sub-optimal result at any time. For example, ref. [5] proposed that for the master-slave architecture introduced earlier (Fig. 3), after the i-th stage the master node gathered the optimisation results $f_{i,y}$ from containers $S_{i,y}$ and decided if the continuation of the optimisation process, i.e. triggering the next optimisation stage, was likely to be beneficial considering the given value curve. A similar approach is applied in this paper.

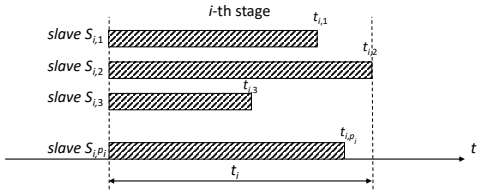


Fig. 4. Stage execution time example

Performing the optimisation at the edge is assumed to cost nothing in terms of money as the edge devices are owned by the smart factory and their idle time can be viewed as wasted. This is in contrast to the optimisation cost in a cloud, which for any i-th stage is nonzero and upperbounded with $\beta \cdot t_i \cdot p_i$, where β is the cost of a single container execution per one time unit[2], p_i is the number of slaves executed at the i-th stage and $t_i = \max_{y \in \{1,\ldots,p_i\}}(t_{i,y})$ (see Fig. 4). The execution cost in both these locations can be described with equation

$$c_i = \Delta_i \cdot (\beta \cdot t_i \cdot p_i), \tag{1}$$

where Δ_i equals 1 if the i-th stage is executed in a cloud or 0 otherwise. Using these notations, the manufacturing profit yielded by the best solution found in the i-the stage is described with equation

$$P_i = VC\left(\sum_{j=1}^{i} t_j\right) - \sum_{j=1}^{i} c_j - f_i, \tag{2}$$

where $f_i = \min_{y \in \{1,\ldots,p_i\}}(f_{i,y})$.

After finishing the optimisation process at stage i, the values of t_{i+1} and f_{i+1} can be predicted via extrapolation, for example using the Bluirsch and Stoer algorithm [7]. For history lengths of 3 or less, such extrapolation is either undefined or else the result was empirically determined to be inaccurate: the predicted value of f_i is then given by the best fitness found so far and that of t_i by the last (actual) processing time.

If the following, $(i+1)$-th stage is processed at the edge, value $\hat{te}_{i+1}$ predicts its execution time and $\hat{f}_{i+1}$ predicts the lowest value returned by the slaves. Both these values can be used to predict the profit generated at the edge after the subsequent stage as follows

$$\hat{Pe}_{i+1} = VC\left(\sum_{j=1}^{i} t_j + \hat{te}_{i+1}\right) - \hat{f}_{i+1} - \sum_{j=1}^{i} c_j. \tag{3}$$

Let us assume that at the i-th stage, executed at the edge, the longest computation (lasting t_i) has been performed by the x-th node with *calibration coefficient* ζ_x and whose fraction of CPU time for the related container equals $\vartheta_{i,y,x}$. This container is predicted to be executed for $\hat{te}_{i+1}$ in the following stage if executed at the edge. Then the execution time in a cloud of the same container can be assessed with formula

[2]For example $\beta = 0.000017$ USD per second of execution per GB of memory allocated using IBM Functions in August 2018.

$$\hat{tc}_{i+1} = \frac{\hat{te}_{i+1}}{\zeta_x \cdot \vartheta_{i,y,x}} \tag{4}$$

Then the profit generated after the subsequent stage executed in a cloud can be estimated with equation

$$\hat{Pc}_{i+1} = VC\left(\sum_{j=1}^{i} t_j + \hat{tc}_{i+1}\right) - \hat{f}_{i+1} - \sum_{j=1}^{i} c_j - \hat{c}_{i+1}. \tag{5}$$

If the current, i-th stage is executed in a cloud, the execution time of the following stage at the edge can be assessed with equation

$$\hat{te}_{i+1} = \hat{tc}_{i+1} \cdot \zeta_x \cdot \vartheta_{i,y,x}, \tag{6}$$

and substituted to equation (3) to estimate the corresponding profit. Value $\hat{tc}_{i+1}$ is also used to estimate $\hat{c}_{i+1}$ using equation (1).

The stopping criteria are evaluated by the master node after each stage i. The *predicted profit* criterion checks the prediction if the execution of the subsequent stage is likely to increase the profit generated by the optimised process or not, regardless it is executed in a cloud or at the edge

$$P_i > \max(\hat{Pe}_{i+1}, \hat{Pc}_{i+1}). \tag{7}$$

Moreover, if $\hat{Pe}_{i+1} \geq \hat{Pc}_{i+1}$, the following stage should be executed at the edge. Otherwise, the containers shall be executed in a cloud.

The benefits of similar stopping criteria in a cloud environment has previously been evaluated [5]. In the following section, we apply this approach to a platform consisting of both edge machines and a cloud.

IV. Experimental results

The edge execution platform described above has been implemented and used together with the original Docker engine in form of two software modules, namely *DockerManager* and *DockerWorker*. The former one corresponds to the master node and is run on a machine where Docker may or may not be installed, whereas the latter, executing the slave nodes, requires the presence of the Docker daemon. These modules are depicted in Fig. 1.

In order to evaluate the technique described in this paper, 30 manufacturing orders considered in ref. [5] have been selected to be scheduled in a certain factory. In that factory, there are 8 machine types and each machine can operate in different operating modes, influencing both the processing time and the consumed energy, which in turn influences the manufacturing cost. The number of manufacturing process steps in these orders ranged from 18 to 59. Each of these steps needs to be allocated to a machine operating in a certain mode. The parameters for the associated value curve are $AT = 0$, $D = 250$ s, $Z = 500$ s and $V_{max} = 5000$ GBP, which means that such amount of money would be gained by a plant if both the production and the scheduling cost nothing.

In the first experiment, the migration between cloud and edge computation has been disabled and both these environments have been used for the first stage. The computation

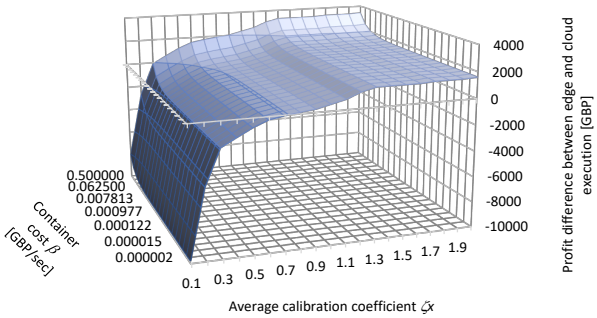


Fig. 5. Profit difference between edge and cloud execution for various average calibration coefficient values and container execution costs

is then performed using the same platform, edge or cloud, from the first stage up to the computation completion. The average calibration coefficient ζ_x ranges from 0.1 (response time from the edge is 10 times longer than from cloud) to 2.0 (response time from the edge is two times faster than from the cloud), and the container execution cost per second, β, varied from 0.000001 GBP to 0.5 GBP. For each setting, the experiment has been conducted 100 times, 400000 runs in total. The difference between the total profit computed in the edge and in the cloud are presented in Fig. 5. Not surprisingly, the time needed for optimisation in case of slow edge computers (i.e. with low average calibration coefficient ζ_x) causes that the computation is usually finished at the time when the associated value curve assumes low values. For extreme case of edge machines with, on average, 10 times lower response time than a cloud (average ζ_x equals 0.1), even assuming the most expensive cloud computation cost (β equals 0.5GBP) leads to high total differences in profits (close to 10000 GBP for the considered set of orders). However, with the increase of the edge machine speed, this difference changes significantly. Assuming typical cloud computation cost in 2018 $\beta = 0.000015$ GBP, edge computations becomes slightly more beneficial (107 GBP difference) for average ζ_x equal to 0.8. For the fastest edge computers considered, with a response time twice as fast as cloud computers, this difference is equal to almost 1600 GBP. As a similar value is achieved even for much cheaper cloud computation ($\beta = 0.000001$ GBP), this proportion will hold even after the forseeable significant decrease in cloud computation cost. In total, processing in edge returned above 8% higher profit than computation in a cloud.

In the next experiment, the slave container migration between edge and cloud is permitted. The computation starts at the edge but migrates to a cloud if the predicted profit of the next stage computed in a cloud is higher than its equivalent predicted for the edge. In the analysed range, the number of migrations from the edge to cloud depends strongly on the ζ_x parameter and to a lower degree on β. For the slowest cloud ($\zeta_x = 0.1$), the migration from the edge to the cloud has been performed in 51% of cases on average, and then decreases almost linearly to 13% for $\zeta_x = 1.0$, i.e. when both the edge and cloud have the same response time on average. For faster edge ($\zeta_x > 1.0$), not a single migration has been observed. For all the considered cases, the possibility of migration to cloud improved the profit slightly, yielding 1% above the execution in edge and 9% higher profit than computation in a cloud. However, this option is more beneficial in case of slow (or busy) edge. For the slowest case ($\zeta_x = 0.1$), computation performed solely at the edge yields 22% worse result than a cloud, whereas the possibility of migrations decreases this gap up to 14%. The migration option may be then viewed as quite beneficial in adverse situations, which remains unused in case of a higher computational power available at the edge.

V. Conclusion

This article describes a distributed architecture that provides general and scalable support for the 'Just in Time' manufacturing process envisioned for smart factories. The architecture is equipped with a novel adaptive stopping criterion for optimising profit obtained from a set of manufacturing orders which not only decides on the computation termination but also steers the computation migration between the edge and cloud. As the optimisation engine has been encapsulated into a stateless container, such migration is occupied with a minimal overhead. According to the experimental results, optimisation at the edge leads to a slightly better overall profit, and in case of slow or busy edge computers, the possibility of container migration to cloud decreases the computation speed gap between a cloud and edge. Since using the proposed approach leads to comparable if not better profits than optimisation solely in a cloud, considering the additional benefits form edge execution, such as reduction of outbound/inbound network traffic, increased reliability and security, edge platform can be viewed as a promising alternative to cloud computing even for computationally costly tasks.

Acknowledgement

The authors acknowledge the support of the EU H2020 SAFIRE project (Ref. 723634).

References

[1] D. Tamburini. (2018) Enabling smart manufacturing with edge computing. [Online]. Available: https://azure.microsoft.com/en-gb/blog/enabling-smart-manufacturing-with-edge-computing/

[2] M. Carrie, V. Turner, R. Yesner, J. Feblowitz, K. Knickle, L. Lamy, M. Xiang, A. Siviero, and M. Cansfield. (2016) Idc futurescape: Worldwide internet of things 2016 predictions. [Online]. Available: https://www.idc.com/research/viewtoc.jsp?containerId=259856

[3] H. El-Sayed, S. Sankar, M. Prasad, D. Puthal, A. Gupta, M. Mohanty, and C. Lin, "Edge of things: The big picture on the integration of edge, iot and the cloud in a distributed computing environment," *IEEE Access*, vol. 6, pp. 1706–1717, 2018.

[4] B. I. Ismail, E. M. Goortani, M. B. A. Karim, W. M. Tat, S. Setapa, J. Y. Luke, and O. H. Hoe, "Evaluation of docker as edge computing platform," in *2015 IEEE Conference on Open Systems (ICOS)*, Aug 2015, pp. 130–135.

[5] P. Dziurzanski, J. Swan, and L. S. Indrusiak, "Value-based manufacturing optimisation in serverless clouds for industry 4.0," in *Proceedings of the Genetic and Evolutionary Computation Conference*, ser. GECCO '18. New York, NY, USA: ACM, 2018, pp. 1222–1229. [Online]. Available: http://doi.acm.org/10.1145/3205455.3205501

[6] P. Dziurzanski and L. S. Indrusiak, "Value-based allocation of docker containers," in *2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, March 2018, pp. 358–362.

[7] J. Stoer, R. Bartels, W. Gautschi, R. Bulirsch, and C. Witzgall, *Introduction to Numerical Analysis*, ser. Texts in Applied Mathematics. Springer New York, 2002.

Network Slicing as an Ad-Hoc Service: Opportunities and Challenges in Enabling User-Driven Resource Management in 5G

Madhumitha Harishankar, Patrick Tague, Carlee Joe-Wong

Electrical and Computer Engineering Department, Carnegie Mellon University
{*mharisha,tague,cjoewong*}*@andrew.cmu.edu*

***Abstract*— Creation of virtualized network instances, aka network slicing, is one of the fundamental architectural paradigms that allows 5G to meet widely diverse service requirements. Recent work has shown the benefits in dynamic rather than static allocation of physical resources to these virtual slices as well as the feasibility of creating and enforcing slices dynamically even in the radio access network. Encouraged by this, we make a case for offering network slicing as a real-time service to end users. This is a significant shift from the predominant business model wherein content providers own these slices and end-users are subject to the resource allocation policy of content providers. By instead enabling users to customize and acquire resources for their desired session performance spontaneously, real-time network virtualization as a service allows diverse applications to drive their network allocation. This is particularly useful for real-time applications with immediate and potentially time-varying resource needs such as predominant use-cases of cyber-physical systems and edge computing. The operator's centralized control of resources is reduced and thereby also the burden of meeting sufficient traffic demand in a slice under sparse availability. Instead, this burden is shifted to the design of incentive schemes that allocate these ad-hoc dynamic virtual slices of limited physical resources to users/devices/applications that value them the most. While enabling diverse service requirements in a decentralized fashion, this brings up new challenges and opportunities in designing mechanisms that capture the wireless end device's spontaneous session needs and valuations.**

I. Network Slicing - Current Model and Concerns

One of 5G's key architectural innovations, network slicing, results from the aggressive network function virtualization that enables complete programmability of network components [1]. Even radio resource allocation and scheduling policies, previously coupled with the physical hardware in the base station, are virtualized and hence programmable. As a result, the network may be divided into virtual slices that potentially span resources all the way from the edge to the core of the network and are dedicated to satisfying demands of a specific service level. This enables 5G's vision of supporting the highly diversified network needs of existing and emerging applications constituting cyber-physical systems (CPS). Machine to machine scenarios (e.g., tactile internet [2], telepresence [3]) that require low latency, high bandwidth and high reliability simultaneously are integral CPS use-cases, as are smart-city scenarios [4] that require periodic transmission of IoT data to the cloud and low-latency computing resources at the edge for making real-time actuation decisions. Simultaneously, the end users' diversity in network requirements and volume of data use also grows.

Virtual network slices catering to different service level agreements (SLAs) emerge as the solution. The network enforces these SLAs by allocating sufficient physical resources to a slice to satisfy the traffic demand [5]. While slicing techniques have been explored extensively (in the mobile core [6], [7], [8] and to some level in the radio access network (RAN) [9], [10]), the *usage models* have not. Simply put, how should end users/devices/applications (collectively referred to as *edge entities* going forward) attempt to acquire guaranteed service for themselves? A B2B (business-to-business) model is predominantly assumed today, wherein content providers work with the operators to define and reserve slices specific to the resource needs of their content, and thereby influence edge entities' quality of experience (QoE) for their applications [11], [12]. However, this model has limiting consequences for *slice usability and utility for edge entities* as well as *slicing efficiency and operating costs*.

Usability concerns: Offering slices as long-term contracts (whether spanning hours or months) to internet content providers or other types of service owners retains the model of centralized resource control and renders a large part of the diverse requirements and use cases of edge entities unfulfilled. Previously at the mercy of the resource allocation policy of a centralized network operator, now edge entities and their network performances rely on the resource allocation policy of the corresponding content-provider/service-owner. This can be severely limiting. First, the edge entity cannot acquire any SLAs for services it values if the service owners have not acquired a dedicated slice from the network, potentially also harboring net-neutrality issues. For example, a user uploading a business-critical document to a server within a tight deadline foreseeably does not have a specific content provider that can allocate a guaranteed slice to complete the task in time. Similarly, in emerging deployments of dense sensors in smart buildings and offices, the sensing systems might have different services to send their periodic data to [4] and may not be able to fully anticipate their future network needs that depend on these services.

Even in the few slicing architectures that account for a B2C (business-to-customer) model alongside B2B, static contracts between the customer and the operator for slices

This work was supported in part by the Defense Advanced Research Projects Agency (DARPA), under contract No. HR001117C0048.

are assumed [11]. Alternatively, content-provider sponsored slices are assumed to exist in which case the architecture tackles the problem of slice discovery and association [13]. This system is hence of *limited use* to a large portion of the heterogeneous edge entities to whom guaranteed SLAs would be expected to add significant value. Due to reliance on service owners to procure and offer these, or the infeasibility of accurately forecasting their changing resource needs that may be ad-hoc and temporal [14], they realize limited utility from slicing. The failure of these relatively static B2C models to leverage the full power of the network's dynamic virtualization capability is alluded to by Zhang et al. [15].

Second, a central content provider or network operator does not know how to *prioritize among its users*. For instance, a provider like Skype contracts a slice that delivers low latency and high bandwidth. This enables high-quality Skype calls for users admitted to the slice but physical resources within the slice continue to be limited. Presumably, when multiple users make Skype calls at congested times, they compete for admission into the slice and have no way to influence the outcome, just as is the case today. For example, a user cannot demonstrate to the slice allocation algorithm their higher call *utility value* for a job interview over a recreational call from another user. **Hence, while admission into a slice largely guarantees slice-specified SLA, admittance itself is entirely controlled by the centralized operator or the content provider in any case.**

Efficiency concerns: Apart from the usability limitations in the centralized slice-ownership model that do not cater to diverse CPS use cases, severe implications on resource utilization have been recently studied [16]. By way of this slicing model, *traffic multiplexing capacity is significantly diminished in the network*. Resources of a slice may be multiplexed only between the traffic demand for that slice rather than between all traffic demand over all resources, as possible today under ad-hoc resource provisioning [5]. Given the inherently sparse nature of radio and other resources in the edge, these utilization losses are highly undesirable.

The lowered utilization efficiency in slicing is empirically analyzed by Marquez et al. [16]. The imposition of a *guaranteed time fraction* in advance as part of slice specifications, i.e., a guarantee that at least a certain percentage of all traffic demand for that slice will be served satisfactorily over fixed time windows, has a steep cost for the operator. Due to the spiky nature of traffic demand, especially closer to the edge, the provider must provision for peak demand within these time windows to ensure the guaranteed time fraction, leading to efficiency loss as high as 80%.

Relaxation of this time window or time fraction does not help beyond a certain extent[16]; efficiency loss may only further be improved if slices acquire *frequent reconfigurability*. For example, if physical resource allocation to slices can be reconfigured every 30 minutes, then efficiency loss is reduced to a best-case scenario of about 20%. Beyond this, the unavoidable effect of multiplexing loss inherent in slicing dominates with no substantial further improvements. As the authors note, given the expenses of operating the network infrastructure and operationalizing such virtualized capabilities, such high resource utilization losses may prohibit monetary feasibility of realizing this model.

These usability and economic viability concerns are implicitly addressed in the model we propose. In this work, we introduce the concept of offering **slicing as an ad-hoc service to edge entities** who, consequently, drive their resource allocation and thereby the QoE for their network sessions. As we detail in the next section, creating network slices in response to edge entities when they express specific SLA needs alleviates the multiplexing loss from up-ahead resource dedication to slices. It creates new opportunities for monetization of value-added services to the network while empowering edge entities and adding value to them.

II. Slicing as a Service to Edge Entities

As Andrews et al. point out [17], network virtualization in 5G revives a radical concept that first emerged in the 1990s: "the provision of user-controlled management in network elements". By offering network slicing as a service in real time to edge entities instead of exposing it via periodic contracts to content providers, 1) current usability and efficiency issues are addressed and 2) new incentive challenges are introduced.

We first note that this proposed shift in the service model of network slicing has significant impact on utilization efficiency. In their empirical evaluation, Marquez et al. [16] show that efficiency loss with slicing becomes negligible only when there are a minimal number of slices (i.e., one dedicated to high-volume, SLA-driven traffic and one mainly serving low-volume, SLA-free traffic) with frequent reconfiguration of assigned physical resources to the slices. In such a scenario, statistical multiplexing gains are largely re-captured, since they are higher when done over a larger portion of the physical resources and traffic. This is best done by maintaining a limited number (if any) of larger slices (perhaps two as above), and instead largely deciding the SLA feasiblily of a flow and its requisite physical resources in real time. A slice is created ad-hoc if the flow with its customized SLA is deemed feasible and sufficient physical resources dedicated to ensure performance isolation. In-fact, recent work [9], [10] has shown that slices can be created and their isolation enforced dynamically even in the highly contended RAN layer while simultaneously maintaining high radio efficiency, making our proposed model feasible.

Secondly, such an offering allows entities with diverse needs (and uncertainty or variation in their future resource requirements, as in several CPS scenarios) to use the system. Further, virtualization shields the edge entity from complex radio-layer details and predictions in computing the resources they need. Edge entities may simply relay requisite network service in terms of application-layer needs (such as bitrate and latency) and have operator compute the mapping to physical resources. Hence any application may procure its session needs, whatever they may be.

In the earlier Skype example, however, we see that the resource needs of a user are not simply the application

or device's inherent network requirements; users also hold subjective preferences and relative utilities for network sessions. These subjective utilities are especially important to capture when edge resources are limited and not every slice request can be met. To enable users to drive their resource allocation, it is not only important to offer suitable slices (or equivalents) in response to their diverse slice specifications, but also allow them to influence its successful allocation by expressing their utility for it. The burden is then placed on the operator to design an effective incentive scheme that aligns the entity's stated utility with its true valuation. The operator may now monetize its virtualization capability by offering it as a real-time value-added service to edge entities, while retaining much of its desirable statistical multiplexing capability. Hence, we realize **the fundamental shift of control from the operator/content-provider to the edge entity, which now drives its network resource allocation in real time as aligned with its incentives.**

III. The Role of Incentives

Historically, edge entities have had limited scope to express the utility they derive from network resources to operators. Prevalent mobile data plans are month-long contracts that do not capture finer-grained information about user preferences and utilities. The current trend towards high diversification of applications and the network services they require, combined with empirical observations that heavy users of cellular internet exhibit non-periodic, sporadic usage [18], indicates that this lack of fine-grained information likely induces a considerable loss in value for both the operator and edge entities. We posit that **offering network slices as a service in real time re-captures this source of value** by allowing operators to offer, and users to pay for, services customized to spontaneous user needs.

Capturing end-user preferences in the form of their utility or valuation and using this to drive resource allocation models is the aim of incentive design mechanisms. The seminal work on Paris Metro Pricing [19], for example, allows users to state their value for the supported QoS levels by explicitly choosing their tier and paying accordingly, assuming that the network can guarantee these tiered performances to all users who pay the price. Since then, several incentive mechanisms for network usage have been proposed with varying goals [20]. However, there has not been significant attention on the problem of capturing user (or generic entity, with the rise of CPS) valuations in real time for application-oriented session preferences. Until now, it has been difficult to realize dynamic QoS policies in today's network architecture with limited flexibility. However, 5G's virtualization features makes such a paradigm entirely feasible. We turn our attention to several open challenges in designing these incentive mechanisms and, more broadly, in realizing a fully functional offering of slicing as a service to edge entities.

IV. Research Challenges

Incentive design: The operator must benefit monetarily by offerring virtualization capabilities to edge entities. Since slice specifications and their resource costs are hereby known only in real time, the operator must price slices in real time. On the other hand, dynamic pricing schemes have been known to have usability limitations [21] since typical end users are budget-constrained and make economic choices over longer time spans. *Building user-friendly mechanisms for dynamic pricing that provably incentivize users to state their true valuations for customized slice specifications is an open problem*. In fact, the most user-friendly method might be to develop agents that act as a proxy for the edge entity in engagements with the network. Once the entity's preferences are captured by the agent, such as budget, applications for which service guarantees are desired, and preferred resolution rates, the agent may transparently engage with the network to acquire a guaranteed slice. The design of such agents and the various learning tasks they may have to perform (see below) poses questions.

Slice Parameterization: Edge entities must state their desired SLAs when requesting guarantees. Since network requirements for a session are best known by the corresponding application (rather than the user), we presume that the application would transparently engage with an agent on the phone to convey requisite bandwidth/latency. These SLAs must also capture user-facing parameters such as preferred resolution and duration and location of consumption. Rather than the entity explicitly engaging with the network to specify this for each network session (thereby potentially degrading usability), agents may instead learn from the entity's preferences and usage patterns to estimate these parameters. Depending on the incentive scheme being employed by the network, the agent may also need to estimate the entity's valuation for the slice and corresponding utility-optimizing slice specifications in real time.

Cost of Dynamicity: Since CPS traffic is expected to be largely machine driven [2], establishing slices in real time would likely involve complex and frequent communication of the requirements between the various edge entities and the network. While recent work [9], [10] has established the feasibility of dynamic RAN slicing, further work is required in studying the signaling overhead/stress caused by slice requests. Further, the turnaround times for resolution of dynamic slice requests must be minimal to facilitate ad-hoc sessions. This requires further study of the delays incurred by the incentive mechanism in determining allocations.

Slice Policies: The operator may wish to enforce generic slice policies to allow opportunities for other edge entities to acquire slices. For example, by enforcing a slice occupation duration of no more than thirty minutes, the limited edge resources are guaranteed to free up periodically for occupancy by other entities. This may also influence the operator's revenue and incentive mechanism, as the market rates for resources presumably change under congested times. However, such policies involve tradeoffs. For example, periodically terminating slices may expose the operator to the risk of decreased market rates during the subsequent time period as demand drops off.

V. Our Preliminary and Ongoing Work

We currently explore incentives for dynamic slice offerings in the context of real-time applications. Sessions of real-time applications are especially hard to provision for due to their immediate resource needs that do not lend to buffer-based adaptations. We enable such applications to acquire guaranteed QoE for their sessions by negotiating with the operator for a slice of their desired SLA.

We first introduce the slice model considered by Marquez et al. [16], where a slice is fully (pre-)specified as $z = (f, w)$, where f and w are such that the operator satisfies at least a fraction f of the slice demand, averaged over discrete time windows w. The operator provisions for peak demand to satisfy f, thereby necessitating additional deployment of physical resources and considerable economic strain. In our entity-driven slice model, a requested slice or SLA may be fully characterized as $z_t = (s(t,d), c)$, where $s(t,d)$ is the dynamic slice specification of the edge entity for consumption duration d at time t and c is the cost of the slice. A slice s is entirely feasible only if it complies with operator's policies and resource availability. The slice cost c may or may not be conveyed by the network (for example, prices may not be explicitly set in an auction bidding scenario). As seen in this model, the operator only needs to decide if enforcing s is feasible, without the burden of guaranteeing the minimum traffic fulfillment f. Instead the burden of fulfillment is offloaded to the incentive scheme. If the entity's valuation is sufficiently high, its traffic is served by the requested slice, otherwise it is turned away.

Based on this, we provide a combinatorial auction mechanism for edge entities to compete for slice allocation in periodic real-time auctions that maximize social welfare. By exploiting the nature of real-time applications, we achieve auctions with winner determinations that are simultaneously fast and incentive compatible, properties not readily achieved in this setting. An agent, on behalf of the edge entity, submits a bid b for z_t while adhering to the entity's daily budget, thereby addressing usability concerns of dynamic pricing models. We also explore learning mechanisms for the agent to learn the edge entities' QoE preferences and place bids proportionally, making the slice procurement process more transparent and user friendly.

VI. Conclusion

We address the diversified service requirements for 5G networks and the resource scarcity that characterizes the network edge through consideration of user-driven mechanisms for real-time resource management. We propose to offer these limited resources as virtualized network slices tailored for the network needs and aligned with the incentives of edge entities. We highlight the promising and feasible directions for such dynamic and ad-hoc slicing, and the efficiency and usability gains that can be achieved using a user-driven approach. We outline the research challenges in realizing slicing offerings, with a focus on incentive design and usability. Finally, we provide a brief overview of our ongoing work in this space.

References

[1] M. Chiosi, D. Clarke, P. Willis, A. Reid, J. Feger, M. Bugenhagen, W. Khan, M. Fargano, C. Cui, H. Deng, *et al.*, "Network functions virtualisation: An introduction, benefits, enablers, challenges and call for action," in *SDN and OpenFlow World Congress*, vol. 48, sn, 2012.

[2] G. P. Fettweis, "The tactile internet: Applications and challenges," *IEEE Vehicular Technology Magazine*, vol. 9, no. 1, pp. 64–70, 2014.

[3] B. Kang, I. Hwang, J. Lee, S. Lee, T. Lee, Y. Chang, and M. K. Lee, "My being to your place, your being to my place: Co-present robotic avatars create illusion of living together," in *Proc. 16th Annual International Conference on Mobile Systems, Applications, and Services*, pp. 54–67, ACM, 2018.

[4] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, "Internet of things for smart cities," *IEEE Internet of Things journal*, vol. 1, no. 1, pp. 22–32, 2014.

[5] X. Foukas, G. Patounas, A. Elmokashfi, and M. K. Marina, "Network slicing in 5g: Survey and challenges," *IEEE Communications Magazine*, vol. 55, no. 5, pp. 94–100, 2017.

[6] X. Foukas, N. Nikaein, M. M. Kassem, M. K. Marina, and K. Kontovasilis, "Flexran: A flexible and programmable platform for software-defined radio access networks," in *Proc. 12th International Conference on emerging Networking EXperiments and Technologies*, pp. 427–441, ACM, 2016.

[7] A. Banerjee, R. Mahindra, K. Sundaresan, S. Kasera, K. Van der Merwe, and S. Rangarajan, "Scaling the lte control-plane for future mobile access," in *Proc. 11th ACM Conference on Emerging Networking Experiments and Technologies*, p. 19, ACM, 2015.

[8] X. Costa-Pérez, J. Swetina, T. Guo, R. Mahindra, and S. Rangarajan, "Radio access network virtualization for future mobile carrier networks," *IEEE Communications Magazine*, vol. 51, no. 7, pp. 27–35, 2013.

[9] X. Foukas, M. K. Marina, and K. Kontovasilis, "Orion: Ran slicing for a flexible and cost-effective multi-service mobile network architecture," in *Proc. 23rd Annual International Conference on Mobile Computing and Networking*, pp. 127–140, ACM, 2017.

[10] A. Ksentini and N. Nikaein, "Toward enforcing network slicing on ran: Flexibility and resources abstraction," *IEEE Communications Magazine*, vol. 55, no. 6, pp. 102–108, 2017.

[11] X. Zhou, R. Li, T. Chen, and H. Zhang, "Network slicing as a service: enabling enterprises' own software-defined cellular networks," *IEEE Communications Magazine*, vol. 54, no. 7, pp. 146–153, 2016.

[12] K. Samdanis, X. Costa-Perez, and V. Sciancalepore, "From network sharing to multi-tenancy: The 5g network slice broker," *IEEE Communications Magazine*, vol. 54, no. 7, pp. 32–39, 2016.

[13] X. An, C. Zhou, R. Trivisonno, R. Guerzoni, A. Kaloxylos, D. Soldani, and A. Hecker, "On end to end network slicing for 5g communication systems," *Transactions on Emerging Telecommunications Technologies*, vol. 28, no. 4, p. e3058, 2017.

[14] C. Marquez, M. Gramaglia, M. Fiore, A. Banchs, C. Ziemlicki, and Z. Smoreda, "Not all apps are created equal: Analysis of spatiotemporal heterogeneity in nationwide mobile service usage," in *Proc. 13th International Conference on emerging Networking EXperiments and Technologies*, pp. 180–186, ACM, 2017.

[15] H. Zhang, N. Liu, X. Chu, K. Long, A.-H. Aghvami, and V. C. Leung, "Network slicing based 5g and future mobile networks: mobility, resource management, and challenges," *IEEE Communications Magazine*, vol. 55, no. 8, pp. 138–145, 2017.

[16] C. Marquez, M. Gramaglia, M. Fiore, A. Banchs, and X. Costa-Perez, "How should i slice my network? a multi-service empirical evaluation of resource sharing efficiency," 2018.

[17] J. G. Andrews, S. Buzzi, W. Choi, S. V. Hanly, A. Lozano, A. C. Soong, and J. C. Zhang, "What will 5g be?," *IEEE Journal on selected areas in communications*, vol. 32, no. 6, pp. 1065–1082, 2014.

[18] U. Paul, A. P. Subramanian, M. M. Buddhikot, and S. R. Das, "Understanding traffic dynamics in cellular data networks," in *IEEE INFOCOM*, pp. 882–890, IEEE, 2011.

[19] A. Odlyzko, "Paris metro pricing for the internet," in *Proc. 1st ACM conference on Electronic commerce*, pp. 140–147, ACM, 1999.

[20] S. Sen, C. Joe-Wong, S. Ha, and M. Chiang, "Pricing data: A look at past proposals, current plans, and future trends," *CoRR, abs/1201.4197*, 2012.

[21] S. Ha, S. Sen, C. Joe-Wong, Y. Im, and M. Chiang, "Tube: time-dependent pricing for mobile data," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 247–258, 2012.

Control over the Edge Cloud - An MPC Example

Karl-Erik Årzen*, Per Skarin*†, William Tärneberg‡, and Maria Kihl‡
* Department of Automatic Control, Lund University, Sweden
†Ericsson Research, Lund, Sweden
‡Department of Electrical and Information Technology, Lund University, Sweden

***Abstract*—The distributed cloud with edge data centers can be used to host mission-critical control applications. In the paper an MPC control approach is used. A test bed that allows physical experiments using a 5G base station, an edge node, and two remote data centers is presented.**

I. Introduction

The future networked society, empowered by hundreds of billions of connected devices, will fundamentally change the way we need and do compute. Already today, the limited computing and storage capacity of end-user and Internet of Thing devices - such as smart phones, laptops, cameras, and sensors - are complemented by remote data centers. However, there are limitations on what kind of interactive and real-time applications that can be deployed on today's cloud due to its inability to provide guaranteed end-to-end performance with low communication delay and little jitter in that delay. A consequence of this is, e.g., that closed-loop control applications with hard real-time constraints currently cannot be deployed in the cloud. One major inhibitor is the delay in excess of 50 ms incurred by today's wireless radio technology, i.e., 4G/LTE. However, with the introduction of 5G, the radio access delay is poised to drop to a few stable milliseconds, removing this hurdle. Still, latency between the base station and the remote data center will typically be too long for many applications. Fog computing is a proposed remedy to this problem. A fog computing infrastructure is hyper-distributed and resource heterogeneous, ranging from user-near data centers at the edge to traditional distant data centers. The edge data centers are typically small data centers associated with a base station, a radio cell, a production plant, some transport system infrastructure, or an office building, see Fig. 1. In the fog, software applications can be dynamically deployed in all types of network nodes to meet their individual performance goals. A prominent use case for edge data centers are latency sensitive closed-loop control applications, e.g., in process control and automation, industrial robotics, or traffic control. Another use case that is actually the driver for the distributed cloud is the ongoing network function virtualization that takes place in the telecommunication network itself. Instead of using dedicated machines to implement different network functions, e.g., firewalls, load-balancers, or intrusion detection devices, one aims to implement them as software modules executing in virtual machines or containers on top of cloud computing infrastructure. The closer the virtualized network functions comes to the radio base band processing the larger the need for local data centers becomes. Once these data centers are in place it is not unlikely that excess capacity will be sold using similar business models as what is used in today's cloud. The contribution of this position paper is the presentation of a use case for edge data centers: mission-critical control over the cloud. Model Predictive Control (MPC) [1] is a control technique that fits naturally for implementation in the cloud. A 5G-based fog computing test-bed is briefly described with which one can conduct real-time experiments with MPC over the cloud including support for dynamic migration of MPC controllers between the different nodes in the distributed cloud using the Internet of Things (IoT) framework Calvin [2]. For more details about the test-bed see [3].

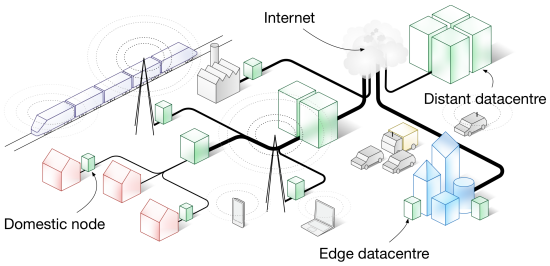


Fig. 1. Distributed fog cloud overview.

II. Distributed Cloud Characteristics

In the distributed cloud compute and storage services are provided in every node from the remote data center to the edge data center in addition to what is provided by the end-user device. The typical control-related characteristics of this scenario are as follows. The closer to the remote data center, i.e., "the further up in the sky", the longer the communication delay will be, the shorter the computational delay will be (more powerful servers), and the larger the capacity will be (more servers). From a control performance point of view deploying the controller in the cloud typically involves a trade-off between the increased communication delay and the decreased computational delay. However, deploying the control computations in the cloud also has other characteristics than the more latency and performance related. An advantage could be the potential access to additional sensor data and more and better models, e.g., obtained using machine learning. A disadvantage could be that the potential points of failure increase and that, due, e.g., to failures or contention, the computations may have to be migrated to some other node in the distributed cloud with different latency characteristics.

III. Control over the Cloud

A natural question to ask is what type of control that may benefit from cloud deployment. In industrial process control the basic control is often provided by Proportional, Integral, and Derivative (PID) controllers. These controllers require very few computations, e.g., around 15-20 lines of C code for a good PID controller including logical safety guard code. Since they typically also guarantee a basic level of performance, safety, and stability it is not recommendable to place this in the cloud. Instead it is the control and planning that is performed at the supervisory control level and above, that are the natural candidates for cloud deployment. In this paper the focus is Model Predictive Control (MPC), a common supervisory controller type. In an MPC the control signal is obtained through the periodic solution of an optimization problem each time new sensor data is available. The advantages of MPC are that it is a multi-input, multi-output (MIMO) controller and that it is able to guarantee that the control signal and the process state fulfill user-defined constraints, e.g., constraints on the maximum and minimum values of the control signals. The disadvantage is that it is compute-intensive and that the execution time can vary substantially, typically more than one order of magnitude, from one invocation of the MPC to the next. This typically has to do with whether the constraints are active or not. The output of the MPC, i.e., the control signals, are used as reference or setpoint signals for the underlying basic controllers. Each invocation of the MPC generates a sequence of control signals, consisting of the control signal(s) to apply at the current sampling instant, at the next sampling instant, etc all the way up to the control signal to apply at the sampling instant given by the current sampling instant plus the horizon. Only the first of these signals is sent to the process and then the same procedure is repeated at the next sampling instant. The input to the MPC is in most cases the process state. Typically an observer, e.g., a Kalman filter, is used to estimate the state vector. Mathematically the MPC controller can be expressed as

$$\underset{u_0,u_1,\ldots}{minimize} \quad \sum_{t=0}^{T-1} L(x_t,u_t)+\phi(x_T) \tag{1}$$

$$s.t. \quad x_{t+1}=f(x_t,u_t) \tag{2}$$

$$u_t \in U, x_t \in X \tag{3}$$

where $L(x_t,u_t)$ is the cost function that should be minimized. The function $\phi(x_T)$ assigns a different value specification to the final (or terminal) state x_T. The number of time steps T is called the prediction horizon and specifies how far into the future the controller anticipates control actions. $f(x_t,u_t)$ is the process model and (3) is a set of expressions which state limitations to the plant inputs and the state space. In the case of a quadratic cost function, a linear process model, and linear constraints, the resulting optimization problem is convex. For convex optimization problems efficient solvers are available. However, the problem with large execution time variations is still present. In the case of, e.g., a nonlinear process model the optimization problem becomes non-convex. Also in this case solvers are available but they in general provide less formal guarantees on optimality etc.

A. Use Case: Process Control

One use case is process control. Here we assume that an MPC controller is used for supervisory control and that it normally is executing in an edge data center and that wireless communication is used. If for some reason connectivity is lost the MPC calculation must be migrated from the edge data center to the local server, i.e., a backup MPC implementation must be available in the local server. This type of migration is an example of a vertical handover. However, due to lower capacity it might not be possible to solve the same MPC problem here. Several possibilities then exist. One is to solve the same optimization problem, but less often, i.e. use a longer sampling period for the MPC. Another possibility is to solve a smaller problem, e.g., only covering the most economically important parts of the plant. When the connectivity is restored the MPC can be migrated back to the edge data center. Another cause for migration could be contention in edge data center or simply a fault in the edge data center. In that case the MPC could either be migrated to the local server as before or be migrated further up, e.g., all the way to the remote data center.

B. Use Case: Fuel Optimization for Trucks

Optimization of fuel consumption for trucks is a problem that can be approached by MPC. Here the aim is to calculate the optimal velocity for the truck that minimizes the fuel consumption taking both geographical map information and dynamic traffic information into account. One could envision that the resulting MPC problem is too complex to be solved completely on an on-board ECU. Hence, the MPC. or part of it, need to placed in the edge data center. However, as the truck moves from one cell to another the MPC needs to follow, i.e., a horizontal handover should be done from the previous edge data center to the edge data center that is now closest to the truck. Also in this use case lost connectivity and poor network coverage need to be handled. In that case the optimization calculations have to be performed in the truck most likely using a simpler formulation.

IV. A 5G-Based Fog Computing Testbed

In order to evaluate the proposed approach a physical test bed has been developed. An overview of the test bed is shown in Fig. 2. The test bed consists of four main parts. As the physical plant to be controlled a ball and beam process is used. The control objective is to move to and keep the ball at a desired reference position on the beam. The ball position and the beam angle are measured using sensors. The mathematical model for the process is a fourth order linear model with the beam angle, beam angular velocity, ball position, and ball velocity as the states. A Raspberry Pi adjacent to the process is used as the local server. It communicates with the cloud via the Lund Massive MIMO (LuMaMi) 5G base station [4], [5]. Massive Multiple Input Multiple Output (MIMO) is the

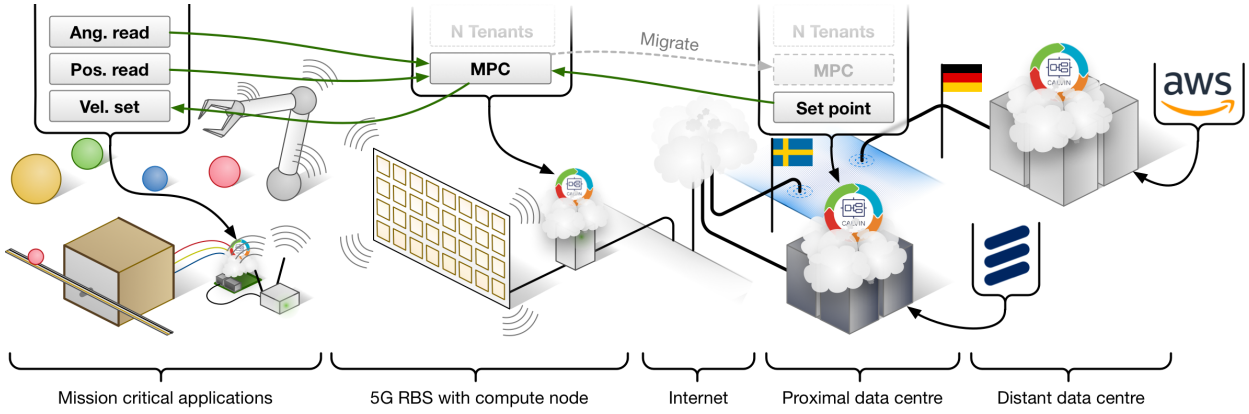


Fig. 2. System overview, from [3]

emerging Radio Access Technology (RAT) for 5G. Fundamentally, massive MIMO is a Multi-User MIMO scheme, which can simultaneously communicate with multiple devices on the same wireless resource. Additionally, massive MIMO operates with significantly more antennas than existing 4G/LTE-based RATs (150 for LuMaMi). Massive MIMO's spectral efficiency is a few orders of magnitude greater than existing RATs. The increased spectral efficiency can be used towards serving more simultaneous devices, increase the throughput, or realizing massive Machine Type Communication where a large number of devices can be reliably served simultaneously at a latency $\leq$ 5ms. A Linux server running with the PREEMPT_RT patch set [6] directly connected to LuMaMi is used to represent the edge data center. The system also consists of the Ericsson Research Data Center (ERDC) situated a few kilometers away from the cell. There we run on top of Open Stack Pike and our instance (a c4m16) has four Intel i7 cores registered by Linux as 1.6 Ghz, and 16 GB of RAM. Finally the system also consist of Amazon Web Services (AWS) data center in Frankfurt where a EC2 instance (a c4.large) with two Intel Xeon cores at 2.9 Ghz and 8 GB of RAM is used.

A. Calvin

As the cloud software platform we use Calvin. Calvin is distributed, event-driven, server-less, and is based on a dataflow programming model. The operational units of Calvin are called actors (nodes in data-flow) while a runtime is an instantiation of the Calvin application environment on a device. In our present implementation there is a one-to-one mapping between Calvin runtimes and compute nodes and we therefore interchangeably refer to them simply as nodes. An actors' input and output messages, are known as tokens. A set of actors and their interconnections constitute an application. Actor states can be migrated and horizontally scaled across nodes.The Calvin framework can autonomously migrate and place actors to load-balance nodes and to meet its own performance goals. However, application owners can specify requirements for actors which tie them to a preferred runtime. For example, a sensor reading actor can be required to be placed on the node associated with the physical plant.

B. Experimental Evaluation

In order to evaluate the approach an MPC controller for the ball and beam process was developed using the QPgen solver [7]. Through Calvin this MPC controller can be migrated between the different nodes in the system. To characterize and verify the basic functionality of the system we ran the MPC on each of the nodes. With each test we let the MPC control the beam for 60 minutes while alternating the set-point of the ball between the centre position and one side of the beam. To be robust in this experiment, we used a large margin to the

Node	RTT (ms)	MPC (ms)	Aggr. delay (ms)
Plant	- - -	5, 5.2, 10	15, 15, 25
Edge	9, 10, 12	0.8, 1.05, 1.2	55, 80, 115
ERDC	12, 13, 18	0.1, 1.1, 1.5	60, 82, 130
AWS	26, 28, 35	0.65, 0.65, 0.7	100, 130, 225

end of the beam, meaning essentially that the likelihood that the constraints would become active was small. The resulting time measurements are shown in the table above. The first column contains the network round-trip time (RTT) from the Raspberry Pi (Plant) to the other nodes. The second column shows the MPC execution time and the third the aggregated latency including both the communication latency, the MPC latency, and the overhead caused by the software platform. Each cell contains three numbers: the 5% value, the median, and the 95% value; all in milliseconds. For a full box plot see [3]. As seen in the table the wireless link introduces a 5ms oneway latency and the Raspberry Pi at the plant is many times slower than the other systems. The AWS node is faster than the edge node and ERDC which is to be expected. It can also be seen that a significant proportion of the delay in the system is introduced by the software platform and not the network. In Fig. 3 we dynamically migrate the MPC randomly across the four nodes. As shown in the figure the process is stable and is able to operate satisfactory in spite

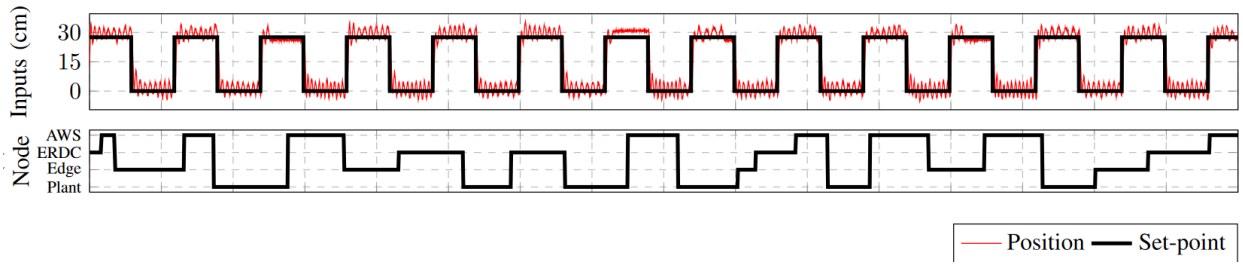


Fig. 3. MPC randomly migrated between the four nodes, from [3]. The top plot shows the setpoint in the black and the ball postion. The lower plot shows in which node the MPC executes.

of the migration and the resulting changes in latency and execution time. However, by tweaking the problem so that the optimization problem becomes more challenging it is possible to have a situation where placing the MPC in the edge node is the only viable approach. If it executes in the locally the execution times becomes too large and if it executes in the AWS the communication latency becomes too large. In both cases the ball falls of the beam. The tweaking of the problem was done by increasing the optimization horizon thus creating a larger optimization problem and by changing the reference signal to be closer to the constraints, i.e. closer to the end of the beam.

V. Timing Issues

In the previous example the same MPC formulation and plant models were used independently of in which node it executed. This MPC formulation ignores the fact that the execution time is non-zero and time varying. However, substantially better results can be achieved if this is taken into account in the design. There are several ways of doing this and here are some examples. Dynamic variations in latency up to one sampling period can be handled by adding a one sample delay to the model. This can be achieved by using a Kalman filter that at time t_k estimates what the state x will be at time t_{k+1}. Rather than waiting with invoking the MPC until time t_{k+1} the MPC is invoked immediately at time t_k. The resulting control signal that arrives at t so that $t_k < t < t_{k+1}$ will then correspond to the control signal that should be applied at time t_{k+1}. By delaying the actuation of this until t_{k+1} latency variations within one sample can be handled. This approach can also be extended to longer delay variations. An approach to handle the slower execution time at the local server could be to increase the sampling period of the MPC, e.g., by a factor 2, when it executes locally and resample the process model accordingly. The rationale behind this is that the sampling period selection rules used in discrete-time control typically provide a range of acceptable sampling periods rather than a single value. Another situation that must be handled is the case when the MPC is running in a cloud node and the control signal is not available at the actuation unit when needed. A possible approach for handling is this is to use the fact that a MPC returns not only the control signal to apply at t_{k+1}, but also the control signal to apply at t_{k+2}. Hence, this signal could be applied to the plant instead, effectively running the process in open loop for one additional sampling period. This is just a few of the techniques that could be used. Several more have been developed in the Networked Control System (NCS) community, e.g., [7].

VI. Conclusions

Using the cloud for deployment of control systems has many attractive advantages. It is possible to use additional information such as high-fidelity models, additional non-local sensor data, learn from the control of other similar plants, and use the latest optimization and learning algorithm. Using this model one could think of control as a service, i.e., Control-As-A-Service (CaaS). However, the requirements on short latency requires the use of local data centers at the edge of the communication network. The migration of the controller between different nodes creates new and challenging control problems related to delay compensation, switching, and mode changes. Model-Predictive Control is a control technique that fits the cloud paradigm well.

VII. Acknowledgements

This work has been partially supported by WASP (Wallenberg AI, Autonomous Systems and Software Program), the Lund Center for Control of Complex Engineering Systems (LCCC), and by the Swedish Research Council.

References

[1] J. Rawlings and D. Mayne, *Model Predictive Control: Theory and Design*. Nob Hill Pub., 2009. [Online]. Available: https://books.google.se/books?id=3_rfQQAACAAJ

[2] P. Persson and O. Angelsmark, "Calvin–merging cloud and IoT," *Procedia Computer Science*, vol. 52, pp. 210–217, 2015.

[3] P. Skarin, W. Tärneberg, K.-E. Årzén, and M. Kihl, "Towards mission-critical control at the edge and over 5G," in *IEEE International Conference on Edge Computing*. IEEE Computer Society, 2018.

[4] J. Vieira, S. Malkowsky, K. Nieman, Z. Miers, N. Kundargi, L. Liu, I. Wong, V. Öwall, O. Edfors, and F. Tufvesson, "A flexible 100-antenna testbed for massive mimo," in *Globecom Workshops (GC Wkshps), 2014*. IEEE, 2014, pp. 287–293.

[5] S. Malkowsky, J. Vieira, L. Liu, P. Harris, K. Nieman, N. Kundargi, I. C. Wong, F. Tufvesson, V. Öwall, and O. Edfors, "The world's first real-time testbed for Massive MIMO: Design, implementation, and validation," *IEEE Access*, vol. 5, pp. 9073–9088, 2017.

[6] Linux Foundation, "Real-time linux," https://wiki.linuxfoundation.org/realtime/start, 2018.

[7] X. Zhang, Q. Han, and X. Yu, "Survey on recent advances in networked control systems," *IEEE Transactions on Industrial Informatics*, vol. 12, no. 5, pp. 1740–1752, Oct 2016.

Machine Learning Enhanced Real-Time Intrusion Detection Using Timing Information

Hang Xu and Frank Mueller, North Carolina State University, USA, hxu9, fmuelle@ncsu.edu
Mithun Acharya and Alok Kucheria, ABB Inc., USA, mithun.acharya, alok.kucheria@us.abb.com

***Abstract*—Past work has investigated intrusion detection mechanisms for real-time control devices. This work contributes a novel framework of separating security monitoring and detection from real-time control, where the former is performed on Cloud edge devices while the latter is run on embedded devices attached to the system that is controlled. We contribute a security monitoring system that validates worst-case timing bounds of the target controller and also validates its control outputs by comparing it against model-based predictions, which are derived from machine learning.**

I. Introduction and Motivation

As industrial and governmental cyber-physical systems (CPS) as well as personal and residential Internet-of-Things (IoT) devices have become ubiquitous, cyber attacks of these systems have also been rising [6]. Such attacks and intrusions usually pose severe risks to the controlled subsystems and may result in home intrusion, dangerous road scenarios, destruction of power grid or other industrial devices that may cause disruptions of operations, bodily injury or even loss of life. The cyber vulnerabilities of CPS and IoT devices lie in the exposure to public networks as required by their functionality and the relatively limited computational capability to deploy heavy-weight traditional security methods, e.g., public key cryptography. These factors motivate our work to significantly increase cyber security across CPS and IoT computing devices with sensitive controls.

II. Past Work and Contribution

In previous work [10], timing the execution paths of and "fingerprinting" the computation tasks in a normal state has been proved effective to identify intrusions on embedded systems. This work assumes the intrusion detection system (IDS) is deployed on the same host as the target CPS or IoT device and is not intruded or can only be intruded after the target is, which may not be the case in practice. In [3], the system states are monitored by a third party FPGA device, which switches into a safe controller mode if the system states are outside of safe thresholds. Such a security system may still be compromised when an attacker tampers with the communication packets to forge a normal state. [1] detects intrusion by monitoring encrypted packets, which cannot be easily deciphered in the communication channel but such encryption may not be feasible on low-end devices. [8] detects intrusion using a physical model but may not be applicable since some control processes cannot be mathematically transformed into models.

We present a novel intrusion detection approach that relies on fine-grained timing information of CPS or IoT devices enhanced by real-time machine learning (ML). **Novel contributions:**

- We separate the IDS from the target embedded system to increase isolation and decrease the attack surface of the detection system. More specifically, the target control system maintains state data, which is transmitted with sensor data to the IDS for verification.
- We use ML models to detect the packets whose data has potentially been tampered with.
- We use ML models for control systems, where the physical system model cannot easily be derived.
- We enhance ML inferencing with tighter and predictable upper bounds on execution time to facilitate a prompt response to intrusions.
- We use highly accurate ML models on an embedded system, exposing it to real industry sensor data from the field, to demonstrates suitability and efficiency for intrusion detection under the constrained resources of embedded systems.

III. A Real-Time Intrusion Detection Framework

Fig. 1 depicts a conceptual overview of our IDS (green/right box), which resides on a different hardware board than the target control system (blue/left box). The IDS communicates with and monitors the streaming telemetries sent from the target control system and external third party sources.

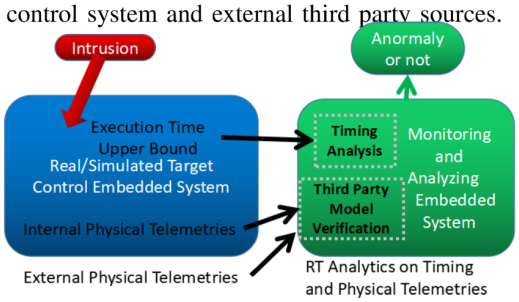


Fig. 1. System Diagram

Three categories of data packets are transmitted from the target control system to the detection system:

- *Execution Time Upper Bound:* We conduct timing analysis on the target control tasks to obtain tight upper bounds on the execution time for specific paths through the control code.
- *Internal Physical Telemetries:* We collect the sensed physical data from the controlled system, which could be the entire or a part of the target control system state.

• *External Physical Telemetries:* We collect the sensed physical data from a third party system with a sensor array and verify the validity of the target control system state.

To make tampering and omitting of the execution time data more difficult, we can anchor the execution time data management in OS kernel as one of the scheduler functionalities. Even if the OS kernel is compromised and the execution times plus system state are tempered with to circumvent detection, simply transmitting arbitrary system state values does not compromise the entire detection mechanism. Detection is not only based on the internal system states but also on their relation to and consistency with external physical states. For example, an intruder may record and replay the output and voltage current of an inverter, yet false power data can still be detected if it does not match the expected values given the current weather data at this geographic location.

Based on the above data, the IDS conducts two analyses, *Timing Analysis* and *Third Party Model Verification*. The former analysis checks the execution time against valid upper bounds obtained from prior experiments. If the error is larger than a certain threshold or the data packets are not received before their deadline, the system will report an anomaly. If the former analysis does not detect any anomaly, the latter analysis is conducted, which supplies the input physical telemetry data to a third party ML model and compares the measured physical telemetry with the expected outcome based on the ML model. The inputs to the model are selected based on the specific control application from internal and external physical telemetries. If the error is larger than a certain threshold, the system will report an anomaly. The pseudo code of intrusion detection is shown in Algorithm 1 using the model parameters of Table I.

The two analyses can be further fused to more effectively counter stealthy attack. A stealthy attack may evade the detection if it results in moderately suspicious execution times and moderately suspicious physical state telemetries. However, the IDS could still detect this attack by considering the execution times and telemetries together, and raising an alarm if both are moderately suspicious. Such fused detection will be based on weighting the thresholds of the timing analyses and the ML verification output and obtaining the summed overall detection threshold. How to weight and fuse the detection thresholds is beyond the scope of this paper. Such fused detection handles independent detection of one data sources as a special case, where other sources receive "zero" weights.

IV. Promptness

Our system addresses one of the most important requirements of IDS, *promptness*, as follows. One of our objectives is to guarantee prompt response to the intrusion through reducing the detection delay, i.e., the duration between the intrusion event and the detection of such intrusion. "Promptness" does not only imply a tight average timing bound of detection delay but also a tight upper timing bound, namely T_{detect}. The total detection delay is the aggregate of two terms:

TABLE I
Parameters of the Detection Algorithm

Symbol	*Description*
N	number of code snippets in the target control code
D_{comm}	deadline of communication delay
$WCET[N]$	worst-case execution time vector
TH_{ML}	ML model verification threshold
$socket$	controller socket file descriptor
$Data$	streaming data from controller
T_{dtc}	the data reception timestamp on the detector
T_{tgt}	the transmission timestamp on the controller
$T_{ctrl}[N]$	execution time vector on target control system
PHY_{in}	internal physical telemetries
PHY_{ex}	external physical telemetries
MSR_{in}	physical telemetries selected as measured inputs
MSR_{out}	physical telemetries selected as measured outputs
EXP_{out}	inference output of ML model

Algorithm 1 Intrusion Detection Algorithm

```
1: function DETECT_ANOMALY( socket )
2:     Data = read(socket);
3:     T_dtc = gettimeofday();
4:     if Data ≤ 0 then
5:         return True;                          ▷ packet not received
6:     else
7:         [T_tgt, T_ctrl[N], PHY_in, PHY_ex] = parse(Data);
8:         if T_dtc − T_tgt > D_comm then
9:             return True;          ▷ data packet not received in time
10:        else if ∃i, T_ctrl[i] > WCET[i], 0 ≤ i < N, i ∈ Z then
11:            return True;                ▷ execution time over bound
12:        else
13:            [MSR_in, MSR_out] = select_telemetry(PHY_in, PHY_ex)
14:            EXP_out = ML_Model(MSR_in)
15:            if ‖EXP_out − MSR_out‖ > TH_ML then
16:                return True;            ▷ ML verification failed
17:            else
18:                return False;                        ▷ No anomaly
19:            end if
20:        end if
21:    end if
22: end function
```

• The duration between the start of the intrusion and the time when the data packets are transmitted to and started to be processed by the detection system, namely T_{comm}.

• The duration between the start and the end of the analysis of the packet data on the detection system, namely T_{analy}.

Reducing the upper bound of the detection delay is not only important to guarantee prompt detection of an anomaly but is also essential to ensure fast data stream monitoring for the target control system.

The transmission interval for streaming data packets can be configured by the programmer to update the execution time bounds and system state to detect an anomaly in a timely manner. Ideally, such an interval is aligned to the control system sampling interval to constantly monitor all system timing bounds and physical states. In order to apply the ideal data transmission interval, the detection delay should be less than the ideal transmission interval. Since the communication delay is determined by the operation system and network medium, we have less margin to tune and optimize the communication delay compared to the analysis delay. Hence,

we use experiments to determine the communication delay and then focus on tightening the upper bound of analysis delays via code optimization. In future work, we plan to enhance the network stack and operating system kernels to further optimize on communication delay.

Communication Delay: Our system uses a communication delay deadline, D_{comm}, on the detection system to verify the validity of the first part of the detection delay, which is the aggregate of T_{proc}^{target} (network processing delay on the control system), T_{trans} (network transmission delay between control and detection systems), and T_{proc}^{detect} (system and network processing delay on the detection system. The network processing delay on both the target and detection systems can be bounded within 2ms [9]. Both systems are usually deployed in the same local area network (LAN) connected via Ethernet with a typical upper bound of delay $<$ 1ms for 100Mb wired Ethernet [2], which is better than any wireless delay [7].

We assume the system clocks are synchronized for the target control and detection systems using Network Time Protocol (NTP) or Precision Time Protocol (PTP) with a system clock error T_{error} bound of 18ms and tens of nanoseconds, respectively [4].

We ensure that the detector waits for and then timestamps the arrival of packets from the controller instead of leaving the arrived packet in the socket buffer and timestamping them later. By not buffering packets, we tighten the upper bound on the communication delay.

Assuming we use NTP and given the above timing bounds, we obtain a theoretical communication delay deadline $D_{comm} = \max T_{comm} + T_{error} = \max(T_{proc}^{target} + T_{trans} + T_{proc}^{detect}) + T_{error} = 2 + 1 + 2 + 18ms = 23ms$.

Analysis Delay: The analysis delay is the worst-case execution time (WCET) of the detection task, which is checking the validity of the execution time upper bound and the state of the control system using the ML model. In our experiment, we found that the WCET for the execution time checking code is significantly smaller than the WCET of the ML model checking code (see experimental section).

The WCET of the ML model checking code is dependent on the ML library deployed on the detection system. Since ML model verification is an inference task, we only consider the WCET of the inference API of the ML library. We select representative ML libraries to compare their suitability for a more predictable upper bound on the execution time of their inference APIs. We select Keras with a Tensorflow backend as the representative for an interpreter-based ML library and Caffe for a compilation-based library. In contrast to the Python interpreter-based Keras library, the inference code of Caffe is written in C++ and compiled into native code, which should in principle result in tighter upper bounds of execution time. Another significant advantage of Caffe over Keras is that it utilizes less memory than Keras and does not dynamically allocate/free any of it. Python's background garbage collector does not provide fine-grained real-time control and often perturbs the predictability of execution time of the ML tasks under Keras. The same is true for Python's reliance on an interpreter, which not only adds overhead for execution but also reduces predictability. (Notice that Python's libraries, such as numpy, often make calls to lower-level C or Cuda libraries for CPUs and GPUs, respectively, which results in better and more predictable performance on higher-end platforms, but not on embedded architectures such as the Raspberry Pi.)

Our experimental comparison shows that the average execution time of Keras's inference phase is about 4 times slower than that of the original Caffe code basis. However, the standard deviation of the execution time, which is directly related to execution time predictability, varies significantly for the original Caffe code distribution, i.e., it is occasionally two orders of magnitude larger and otherwise 4 times smaller than that of Keras. This somewhat surprising result shows that Keras outperforms the original Caffe code in performance and real-time predictability for the ML task of inferencing.

V. Accuracy

The accuracy of our system is evaluated by the confusion matrix, where the false negative (FN) rate indicates undetected anomalies and the false positive (FP) rate indicates normal state flagged as abnormal.

The overall system accuracy is affected by both the accuracy of execution time upper bound checking and the accuracy of ML model verification. We denote the FP rate and FN rate for the timing analysis and ML model verification as FP_{tm}, FP_{ML},FN_{tm} and FN_{ML} respectively. The derivation can be briefly described as follows. An FN detection of the overall system occurs when and only when an attack takes place to the controller but neither timing analysis nor ML verification detect such an intrusion. Thus, an FN event implies both timing analysis and ML verification failed, which is equivalent to multiplying the probabilities of these two independent detectors. In contrast, an FP occurs when there is no attack but either timing analysis or ML verification flag an anomaly, i.e., the union of FP events of the two detection methods. Since an anomaly flagged by timing analysis precedes ML verification, the FP event of the ML verification coincides with a true negative (TN) of timing analysis. Since $TN = 1 - FP$, the overall system detection FN, FN_{sys}, and false-positive, FP_{sys}, rates are:

$$FN_{sys} = FN_{tm} * FN_{ML} \quad (1)$$

$$FP_{sys} = FP_{tm} + (1 - FP_{tm}) * FP_{ML} \quad (2)$$

Here, we observe that FN_{sys} is reduced by a factor of $0 <= FN_{ML} <= 1$. Although an extra term, $(1-FP_{tm})*FP_{ML}$, is added to the overall system FP rate, detection still depends on FN and FP. In other words, a trade-off exits between the cost of reacting to false alarms and missing anomalies, where the latter exposes systems to greater risk. Such a study is beyond the scope of this paper. Instead, we focus on configuring the detection threshold of the timing analysis and the model verification for better overall accuracy of the detection system based on a pre-trained ML model.

VI. Experiment

We consider a practical industrial problem, where a green (solar) power generation source is secured. The core part of

a solar power system is the inverter, which controls power conversion and flow via an embedded micro-controller or DSP. We simulate the solar inverter's embedded system and the intrusion detector on two Raspberry pi B microcomputers, respectively, to assess accuracy and response time of the IDS.

The simulated inverter (controller) is a real-time application with two tasks, one of which reads data from a file to simulate sensing while the other sends packets containing the sensed data and execution time information to the detector. The detector is a real-time application with two tasks, one for reading and parsing the TCP message from the controller and the other for anomaly detection analysis described in pseudo code 1. To allow the anomaly detector to keep up with the data streaming rate of the controller and to consider the WCETs of all tasks on the controller and detector, we choose 25ms as the relative deadline for both tasks on the controller, and 40ms and 10ms for those of the data reading and anomaly detection tasks of the detector, respectively.

Via our partner, ABB Inc., we have access to field data collected from ABB's $UNO\_2.0\_2.5$ inverter and ABB's weather station $VSN800-14$ from 2014 to 2017 located at Kihei, Hawaii. This data set consists of electrical state data from the solar inverter and weather data from the weather station deployed $20km$ away from the solar inverter plant, both collected every five minutes. The electrical data reflects the inverter's output power generation, and the weather data includes the ambient temperature, the global horizontal irradiance (GHI) and plane of array irradiance (POAI) etc. at the weather station. In this experiment, the detector selects the weather information as ML model input, computes the predicted inverter output power as ML inference result and compares the result with the measured output power, which enhances anomaly detection beyond timing bounds checking. We ignore the data collection interval of 5mins and assume it aligns with the task period, 50ms, of the control system, since timing analysis and ML inference are performed offline.

Controller and detector are sharing the same wired LAN of a router to reduce the communication delay. The Raspberry Pis synchronize with the same NTP server. The worst-case communication delay was experimentally upper bounded at 2.7ms, aggregating network transmission delay, OS processing delay considering NTP synchronization error, which is notably much lower than $D_{comm} = 23ms$ (see Sect. V), likely due to different networking than used in previous work [4]. Due to clock drift, time-difference tables, dynamically updated as part of exchanged network packets, may be more suitable here [5]. Our target code snippet of the controller has a measured execution time of 10ms. We simulate additional execution due to malicious code via "sleeping" for by random intervals of $0-10$ms. For reproducibility, we set the seed of the random number generator. Based on the these parameters, we choose the detection threshold of the execution time to be 0.1%, 1%, and 10%, namely 0.01, 0.1 and 1ms. 80% of data samples are subject to intrusion. The other 20% feature timing deviation less than 0.01ms, which are not considered to be intrusions. We offline train our ANN model via the Caffe library with high inference accuracy in terms of variance, namely 0.891. Since the power output of the inverter is below 2kW, we configure the detection threshold for ML inference deviation to be 1%,5%, and 10% of 1kW, namely 10, 50, and 100W.

We conduct 9 groups of experiments with different detection thresholds of execution time and ML model output deviation, each group has 10 experiments with 2500 data samples per experiment. We average over the 10 groups and compare the confusion matrix of the overall detection system combining timing analysis and ML model anomaly detection with sole timing analysis checks (past work).

Fig. 2 shows that the system's FP rate decreases when detection thresholds of the WCET or the ML model increase. However, the FP rate of the overall system tends to increase and the accuracy (true positive plus true negative) tends to decrease when detection thresholds of the WCET or the ML model increase. This illustrates the trade-off between selecting appropriate detection thresholds as a means to optimize overall system efficacy considering the effects of FP and FN events for a specific application.

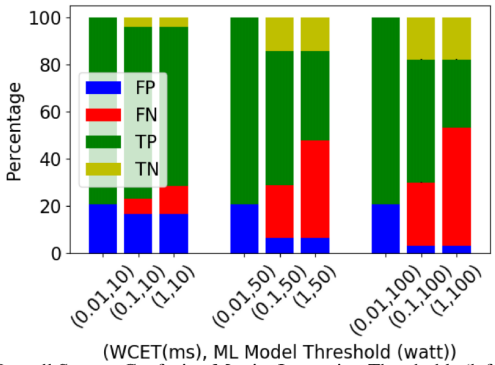


Fig. 2. Overall System Confusion Matrix, Increasing Thresholds (left to right)

Fig. 3 show that ML helps increase detection accuracy of the overall system by about 3%, especially when the execution time upper bound is not as tight (0.1 and 1ms).

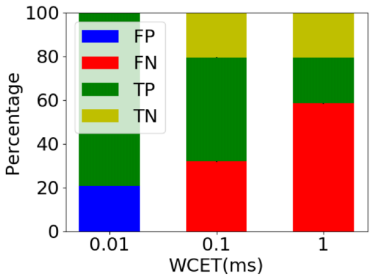


Fig. 3. Timing Analysis Stacked Bar Confusion Matrix

VII. Conclusion

We enhanced prior intrusion detection based on timing analysis via ML model verification and conducted experiments to demonstrate its effectiveness based on practical industrial data. We investigated the trade-off between FP and FN rates when selecting the detection thresholds of WCET and ML model output. Future work will capitalize on this trade-off and develop optimization techniques to further improve intrusion detection for CPS/IoT.

ACKNOWLEDGEMENT

This work was funded in part by NSF grants 1329780, 1813004.

REFERENCES

[1] Sachin P. Joglekar and Stephen R. Tate. Protomon: embedded monitors for cryptographic protocol intrusion detection and prevention. *International Conference on Information Technology: Coding and Computing, 2004. Proceedings. ITCC 2004.*, 1:81–88 Vol.1, 2004.

[2] Ming Li. Delay analysis of networked control systems based on 100 m switched ethernet. *TheScientificWorldJournal*, 2014:751491, 2014.

[3] Sibin Mohan, Stanley Bak, Emiliano Betti, Heechul Yun, Lui Sha, and Marco Caccamo. S3a: Secure system simplex architecture for enhanced security and robustness of cyber-physical systems. In *Proceedings of the 2Nd ACM International Conference on High Confidence Networked Systems*, HiCoNS '13, pages 65–74, New York, NY, USA, 2013. ACM.

[4] T. Neagoe, V. Cristea, and L. Banica. Ntp versus ptp in com puter networks clock synchronization. In *2006 IEEE International Symposium on Industrial Electronics*, volume 1, pages 317–362, July 2006.

[5] Tao Qian, Frank Mueller, and Yufeng Xin. Hybrid edf packet scheduling for real-time distributed systems. In *Euromicro Conference on Real-Time Systems*, pages 37–46, July 2015.

[6] A. Sadeghi, C. Wachsmann, and M. Waidner. Security and privacy challenges in industrial internet of things. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, June 2015.

[7] Shweta Singh and Arun Tripathi. Analysis of delay and load factors in wired and wireless environments, 12 2015.

[8] Nils Svendsen and Stephen Wolthusen. Using physical models for anomaly detection in control systems, 03 2009.

[9] J. Xie and M. Xie. Delay bound analysis in real-time networks with priority scheduling using network calculus. In *2013 IEEE International Conference on Communications (ICC)*, pages 2469–2474, June 2013.

[10] C. Zimmer, B. Bhat, F. Mueller, and S. Mohan. Time-based intrusion dectection in cyber-physical systems. In *International Conference on Cyber-Physical Systems*, pages 109–118, April 2010.

Social Welfare-based Optimization for Data/Service Delivery to Connected Vehicles via Edges

Deepak Gangadharan, Oleg Sokolsky and Insup Lee
University of Pennsylvania, USA
Email:{deepakg,sokolsky,lee}@cis.upenn.edu

BaekGyu Kim
Toyota InfoTechnology Center, USA
bkim@us.toyota-itc.com

***Abstract*—Technologies enabling connected and automated vehicles are increasingly becoming the focus of research and development in the automotive industry. Consequently vehicles are equipped with different modes of connectivity, which enable them to communicate with various entities such as the cloud (V2C), infrastructure (V2I), other vehicles (V2V) and pedestrians (V2P). However, recently vehicular edge computing has become popular, whereby vehicles communicate with edge nodes, which are physically located close to vehicles as it reduces the turn around time compared to V2C. There are various types of data and service that can be provided to the vehicles through the edge nodes, such as content delivery, computation offloading, etc. In this work, we propose a social welfare-based optimization framework for data/service delivery to connected vehicles via the edge nodes while considering the vehicle traffic flow in the edge coverage area. Additionally, we present our initial results on optimally delivering data/service to vehicles on the move, while considering both delivery time and edge bandwidth cost objectives.**

I. Introduction

In the past few years, the automotive industry has focused its research and development on technologies that enable the realization of connected and automated vehicles. This effort has seen lots of innovation in the development of novel networking technologies and applications that enhance driver safety and experience, such as intelligent driving, driver behaviour analysis and predictive maintenance. Many of these applications either receive some kind of data from the external entities such as cloud or are computation intensive functions. Therefore, storage and computation resources are very critical to the efficient execution of these applications. Although it is best to execute these applications on the vehicle platform to minimize latency, it may be computationally infeasible due to the limited capacity of the compute resources available as many concurrently running applications result in resource contention.

One solution to the computation problem has been to offload the execution of an application partially or entirely to the cloud server due to its massive computational capacity. The results of the computation are then sent back to the vehicle. However, this approach results in excessive turnaround time due to communication latency. Additionally, with more vehicles offloading their computation, there will be contention for communication bandwidth. The above two problems are addressed by the paradigm of mobile edge computing (MEC). In MEC, there are edge nodes, which sit in between the cloud and the vehicles and are physically located closer to the vehicles. Any data (such as update data) that is sent from the cloud to a group of vehicles can be sent via the edge nodes, which will result in reduction of bandwidth usage as the same data need not be sent multiple times to different vehicles. The vehicles can then retrieve their data as they pass through the coverage area of the edge. Similarly, computation can be offloaded from the vehicle to the edge instead of the cloud, which considerably reduces the turnaround time.

In the context of edge computing-based data/service delivery to the vehicles, one important question that needs to be answered is how to allocate resources on the edge nodes. The storage, computational and bandwidth capacities of the edge nodes are constrained in comparison to the resource capacities on the cloud. Therefore, it is essential to allocate the edge resources such that the target objective functions are optimized. In this work, we propose a social welfare-based optimization framework to deliver data/services to vehicles on the move while considering the objectives of delivery time and edge bandwidth cost.

In Section II, we present the motivation of our work. Then, we discuss the proposed optimization framework in Section III. The initial experimental results are presented in Section IV. In Section V, we highlight our conclusions.

II. Motivation

Recently, the US Department of Transportation has funded three connected vehicle pilot projects in New York, Wyoming and Tampa [2], where the goal is to evaluate the effectiveness of using an infrastructure consisting of road side units (RSUs) (as edge devices) in improving the driver safety. The pilot projects in New York and Tampa operate in an urban environment whereby the vehicles communicate with the RSUs, other vehicles and traffic lights to improve safety of drivers, pedestrians and to improve the traffic flow through intersections. On the other hand, the pilot project in Wyoming operates in an Interstate route and helps the drivers in situations of bad weather to give alerts and provide safety from crashes. In this work, we leverage this RSU based edge infrastructure to deliver important data/services which improve the driver safety and experience. Hence, we propose an optimization framework to allocate resources on the edge nodes (or RSUs in the case of the pilot projects) such that data/service delivery is accomplished while optimizing some target system objective.

The question of optimal resource allocation on edges in the context of connected vehicles on the move is dependent on the following factors - edge resource capacity, vehicle route, traffic flow density, edge coverage area and the target system objective (delivery time, bandwidth cost, etc.). The higher the resource capacities on edges, more the vehicles that can be served. The vehicle route imposes a constraint on which edges can be used to deliver data/service. If the traffic flow density is too high in the coverage area of an edge, then it may not be feasible to deliver any data/service to a vehicle via that edge. Therefore, in this work, we attempt to capture the effects of traffic flow density on the optimization objectives. Intuitively, the larger the edge coverage area, more is the time each vehicle has to retrieve its data/service from the edge. We consider all these factors in formulating our optimization problem in the Section III.

The research that is closest to our work are the papers on data delivery from infrastructure to vehicles [6] and online resource allocation to deliver services like computation offloading [8] for mobile nodes. In [6], the authors propose a trajectory-based forwarding scheme to deliver data from infrastructure nodes to moving vehicles in vehicular adhoc networks. An online edge cloud resource allocation algorithm is proposed in [8], which considers arbitrary user movement and variation in resource prices. Due to arbitrary user movement, the cloud does not a priori know the route that the vehicle will take. In contrast, it is a very likely scenario that a driver knows beforehand the route to the destination. In such circumstances, the cloud can exploit the route information to deploy data/service, which has been used in [6]. However, both these techniques do not consider any vehicle flow model, which characterizes the movement of traffic near the edge or between edges.

Our prior work [3] on data/service delivery proposed an optimization framework to address the issues of other works mentioned above. The proposed optimization framework in [3] considers the vehicle traffic flow density while allocating resources on the edge for data/service delivery. However, the optimization was performed only considering a bandwidth cost objective. Delivery time of the data/service is also an important system objective. Therefore, in this work, our proposed optimization framework addresses this problem by introducing a *social welfare function* that considers the target requirements of delivery time and bandwidth cost. We optimize the social welfare objective for a group of vehicles entering the system and present some initial results showing the advantages of using the social welfare function.

III. Optimization Framework

The optimization flow starts in the cloud when it receives a bunch of requests from the vehicles in order to either deliver some data or service. Along with the request, the vehicles also send information regarding the route that they intend to take. From the route map, the cloud can infer the edges that the vehicle will pass through. Let us assume that there are N vehicles requesting for either data or service and there are M edges in the network available for data/service delivery. The goal of the optimization step in the cloud is to find appropriate edges in the edge network such that a system objective (for e.g., total bandwidth cost, delivery time, etc.) for delivery of data/service from the edges to the vehicles is optimized.

First, we will describe the terms used in our optimization framework and then we will present a discussion on the constraints and optimization objective. For the data delivery problem, the decision variable is $m_{i,j}$, which is the size of data chunk for vehicle V_i (it is either a part of V_i's entire data or the entire data) that is assigned to edge E_j. We use a term $x_{i,j}$ to indicate that vehicle V_i passes through edge E_j. The memory capacity and memory currently occupied on edge E_j is denoted by M_j and M_j^{occ} respectively. The memory requirement for the data required by vehicle V_i is given by M_i.

A. Optimization Constraints

We will describe the five constraints that are necessary to check for feasible data delivery to the vehicles. The constraints for the service delivery also follow the same idea, which we will discuss briefly but do not present in detail.

1) **Range Constraint**: The upper and lower bound for the decision variable $m_{i,j}$ is given as follows.

$$\begin{aligned} m_{i,j} = 0, i = 1..N, j = 1..M : x_{i,j} = 0 \\ m_{i,j} \geq 0, i = 1..N, j = 1..M : x_{i,j} = 1 \\ m_{i,j} \leq M_i, i = 1..N, j = 1..M : x_{i,j} = 1 \end{aligned} \tag{1}$$

The upper bound for $m_{i,j}$ is the size of the data that the vehicle requests for, while the lower bound is 0.

2) **Accumulation Constraint**: Given that data may be broken into chunks and stored on more than one edge, the summation of the sizes of the data chunks over all the edges should not exceed the data size M_i. This constraint is formulated as shown below.

$$\sum_{j=1}^{M} m_{i,j} \times x_{i,j} = M_i, i = 1..N \tag{2}$$

3) **Time to Edge Constraint**: This timing constraint ensures that a vehicle reaches an edge not before the data chunk has been transferred by the cloud to the edge. We formulate this constraint as follows.

$$m_{i,j} \times t\_comm_{i,j} \leq m_{i,j} \times t\_trv_{i,j}, i = 1..N, j = 1..M \tag{3}$$

where $t\_comm_{i,j}$ is the time taken to send a data chunk $m_{i,j}$ to edge E_j and $t\_trv_{i,j}$ is the time taken for the vehicle V_i to travel to edge E_j after the vehicle V_i initiates the download. We use the decision variable $m_{i,j}$ on both sides of the inequality in order to not map any data chunk to the edges where the timing relationship $t\_comm_{i,j} \leq t\_trv_{i,j}$ is not satisfied.

4) **Edge Resource Constraint**: For data delivery, we consider the memory constraint, where we ensure that the amount of memory required for all the chunks belonging to all the vehicles passing through an edge does not

exceed the memory capacity of the edge. This constraint is formulated as below.

$$\sum_{i=1}^{N} m_{i,j} + M_j^{occ} \leq M_j, j = 1..M \tag{4}$$

5) **Bandwidth Schedulability Constraint**: Given the bandwidth B_j offered by the edge E_j, the amount of data available to a vehicle V_i can be computed based on a macroscopic vehicle flow model in the coverage area of an edge, which is described in [7]. The macroscopic flow model depends on three quantities - vehicle density k_j, vehicle flow q_j and vehicle velocity $v_{i,j}$. The three quantities in the flow model are related as follows

$$q_j = k_j \times v_{i,j}$$

Greenshields [5] proposed a linear relationship between speed and density based on field experiments given by

$$v_{i,j} = v_j^f \times (1 - \frac{k_j}{k_j^{jam}})$$

where v_j^f is the free flow velocity near edge E_j, which is the velocity of the vehicle when it has no obstructions from other vehicles (usually assumed to be the speed limit) and k_j^{jam} is the vehicle density during a jam. The vehicular traffic in the coverage area of the edge can be modelled using a M/D/C/C queuing model for a steady-state traffic flow model described above as shown in [7], whereby the traffic arrival at the edge follows a Poisson process, servicing the traffic is deterministic and there are C servers. If the coverage distance of an edge is L_j, then $C = k_j^{jam} \times L_j$ is the maximum number of vehicles that can be accommodated in the coverage range of the edge. Then the minimum number of bytes received by a vehicle is given by

$$D_{i,j}^{min} = \frac{B_j}{k_j^{jam} \times v_{i,j}} \tag{5}$$

where B_j is the bandwidth of the edge device.

The bandwidth schedulability constraint for vehicle requiring data is given by

$$m_{i,j} \leq D_{i,j}^{min}, i = 1..N, j = 1..M \tag{6}$$

which simply depicts that the data chunk mapped to edge E_j for vehicle V_i must be less than or equal to the minimum number of bytes that V_i will receive from E_j during its journey across the coverage distance. Additionally, if B^{req} is the minimum bandwidth required by other vehicles near edge E_j, then the bandwidth schedulability constraint for other vehicles in the coverage area is given by

$$B^{req} \leq \frac{B_j}{L_j \times k_j^{jam}}, j = 1..M \tag{7}$$

which means that the minimum bandwidth available for a vehicle (given by the right hand side term) should be no less than B^{req}.

In the case of service delivery, there are additional accumulation constraints considering the processing requirement following the same idea as for memory requirement. Similarly, the edge resource constraints also look at both memory constraints and processing capacity constraints. Here the difference is that there is some data sent by the vehicle to the edge and the result data is sent back to the vehicle. In the case of bandwidth schedulability constraint, we need to consider the bandwidth for data transfer from vehicle to edge and back to the vehicle.

B. Objective Function

In this section, we introduce the social welfare objective function that considers both data delivery time and bandwidth cost. In order to capture delivery time, we use the concept of utility function $U(.)$, which ensures that the utility value decreases as the vehicle goes further away from the point of origin where it initiated the data download request. The utility function increases in value with increase in the value of chunk size and saturates after a certain value of chunk size equal to the size of the data because the driver satisfaction beyond that saturates. Thus, utility function $U_{i,j}(.)$ for vehicle V_i on edge E_j is strictly concave and we use a modified version of $\log(.)$ function used in earlier works on network bandwidth allocation as shown in [4].

The bandwidth cost function is captured using a congestion function. The congestion function uses a bandwidth pricing mechanism, which increases the cost of using the bandwidth at an edge if the bandwidth utilization of the edge is already high. When the bandwidth utilization is low, the cost of using the bandwidth is lesser. Traditionally, in network bandwidth allocation, a quadratic function of bandwidth utilization has been used to determine the congestion cost as given below

$$C_j(bw_j^{util}) = \beta \times (1 + bw_j^{util})^2 \tag{8}$$

where β is the cost factor and bw_j^{util} is calculated based on the bandwidth allocated to the vehicles with chunk $m_{i,j}$ to be downloaded and the vehicles with minimum bandwidth requirement of B^{req}.

The social welfare function can then be calculated as

$$S(B_{i,j}) = \sum_{i=1}^{N} \sum_{j=1}^{M} U_{i,j}(B_{i,j}) - \sum_{j=1}^{M} C_j(bw_j^{util}) \tag{9}$$

C. Optimization Frequency and Resource Usage Duration

We plan to extend the framework to also consider the temporal aspect pertaining to more than one optimization, i.e., our current optimization framework only looks at the constraints relevant for one optimization and therefore, we will analyze the effects of time interval between two consecutive optimizations on the data/service delivery optimality. Similarly, in the current work, we do not include constraints that reflect the release of memory once the data/service is

delivered, which is essential for efficient resource allocation for data/service delivery. We plan to include this by releasing the memory after the latest time when the vehicle will pass by the edge.

IV. Preliminary Results

In this section, we demonstrate the advantage of using the social welfare function in comparison to an objective function which only considers the bandwidth/congestion cost. We will first explain the experimental settings and then present our inferences from the results obtained. The data for the experiments were generated using Matlab scripting and the optimization was carried out using the CVX Solver.

The ranges of the parameters used in this experiment are taken from sources such as [7], [1], [8], [2] and [4].

1) Number of edges (M) was fixed to 49.
2) Number of vehicles (N) requiring data was fixed to 120.
3) Vehicle jam density (k_j^{jam}) was randomly generated using uniform distribution between 40 and 50.
4) The actual vehicle density (k_j) in the coverage area was fixed to 35.
5) Coverage distance (L_j) of each edge was assigned arbitrary values between 0.6 miles and 1.6 miles.
6) Memory requirement of data requested by each vehicle (M_i) was randomly generated using uniform distribution between 60 Mbits and 80 Mbits. These values are very much in the range of realistic software update sizes as given in [1].
7) Memory capacity of the edge (M_j) was randomly generated using uniform distribution between 400 Mbits and 500 Mbits.
8) Maximum bandwidth capacity of the edges (B_j) was randomly generated using uniform distribution between 8 Mbps and 15 Mbps.
9) Free flowing velocity (v_j^f) of the vehicles in the coverage area of edges was randomly generated between 50 and 70 mph.
10) The route of the vehicles was randomly generated by picking connected edges randomly from a square grid.
11) The earliest travel times of the vehicles to the edges on its route was generated based on the distance of the edges from each other and free flowing velocity of the vehicles between the edges, both being generated randomly.
12) The bandwidth cost factor β is fixed to 0.36.

The optimization result is shown in Fig. 1, where the delivery times of data is compared between social welfare based optimization (red "plus" plots) with that of bandwidth cost based optimization (green "circle" plots). It is very evident that many vehicles receive their data much earlier using the social welfare objective function. However, there are some vehicles which receive their data later using social welfare objective function. On an average considering 120 vehicles, the optimization using social welfare objective function resulted in earlier data delivery by 91.08 secs. The increase in bandwidth cost was marginal by 0.3742, i.e., the bandwidth cost using

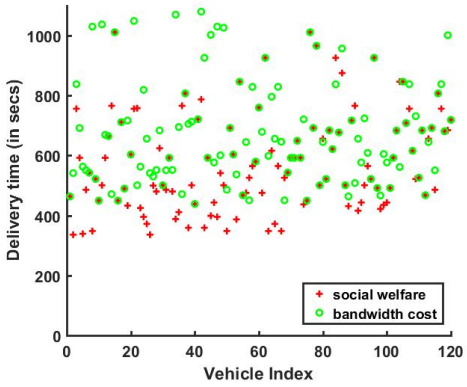


Fig. 1: Comparison of Data Delivery Times between Social Welfare Objective and Bandwidth Cost Objective

social welfare function increased by 0.3742 in comparison to the bandwidth cost function. Therefore, our initial results show that we can perform faster delivery of data to vehicles using a social welfare function that we proposed in this work.

V. Conclusion

In this paper, we present a social welfare-based optimization framework for data/service delivery to connected vehicles via edge devices. We proposed a social welfare function, which takes into consideration the delivery time and bandwidth cost parameters. The optimization results demonstrate that it is possible to deliver data faster to the vehicles via the edges by using the social welfare function. As part of the future work, we plan to conduct more experiments by varying the values of different parameters including the problem size. The social welfare-based optimization is complex and therefore we also intend to work on a heuristic solution for data/service delivery.

References

[1] Updating car software: Why delta technology is better than compression. http://www.embedded-computing.com/embedded-computing-design/updating-car-software-why-delta-technology-is-better-than-compression.

[2] U.S. department of transportation, connected vehicle pilot deployment program. https://www.its.dot.gov/pilots/.

[3] D. Gangadharan, O. Sokolsky, I. Lee, B. Kim, C.-W. Lin, and S. Shiraishi. Bandwidth optimal data/service delivery for connected vehicles via edges. In *11th IEEE International Conference on Cloud Computing (CLOUD)*, pages 106–113, 2018.

[4] A. Ghavami, Z. Li, and H. Shen. On-demand bandwidth pricing for congestion control in core switches in cloud networks. In *9th IEEE International Conference on Cloud Computing (CLOUD)*, pages 867–870, 2016.

[5] B. Greenshields, W. Channing, H. Miller, et al. A study of traffic capacity. In *Highway research board proceedings*, volume 1935. National Research Council (USA), Highway Research Board, 1935.

[6] J. Jeong, S. Guo, Y. Gu, T. He, and D. H. Du. Trajectory-based statistical forwarding for multihop infrastructure-to-vehicle data delivery. *IEEE Transactions on Mobile Computing*, 11(10):1523–1537, 2012.

[7] W. L. Tan, W. C. Lau, O. Yue, and T. H. Hui. Analytical models and performance evaluation of drive-thru internet systems. *IEEE Journal on selected areas in Communications*, 29(1):207–222, 2011.

[8] L. Wang, L. Jiao, J. Li, and M. Mühlhäuser. Online resource allocation for arbitrary user mobility in distributed edge clouds. In *37th IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 1281–1290, 2017.

RRP Edge Computing System

Guangli Dai Pavan Kumar Paluri Albert M. K. Cheng
Department of Computer Science, University of Houston, Houston, TX 77004, USA
email: {gdai, pvpaluri, amcheng}@uh.edu

***Abstract*—Due to the popularity of the Internet of Things, more and more computing resources are needed for smart devices. To relieve the dependency on clouds and to lower the cost of equipment, we propose a Regularity-based Resource Partition Edge Computing System (RRP-ECS). In this system, RRP divides each CPU into 2 partitions, one of which handles the basic control, while the other functions as a core of a virtual machine dealing with intensive tasks. We introduce a centralized structure and discuss the critical task mapping problem in RRP-ECS in detail.**

I. Introduction

As a part of the Internet of Things (IoT), more and more smart devices are deployed in homes and cities these days. However, smart devices require a large amount of computing resources. Dedicated localized computing resources are expensive while dependency on cloud computing leads to inevitable latency. This paper proposes an edge computing system based on the Regularity-based Resource Partition (RRP) model [1].

Considering that smart devices do not have heavy-loaded tasks at all times, equipping a high performance CPU for each smart device would be expensive and wasteful. Therefore, equipping a relatively low-configuration CPU for each smart device and sharing computing resources with each other would be less expensive and more efficient. RRP Edge Computing System (RRP-ECS) is a virtual machine built on multiple smart devices as shown in Fig. 1.

For smart devices, tasks can be divided into two classes: 1) *basic control tasks*, which are fundamental and low-cost, and 2) *high-level intensive tasks*, e.g., analyzing the data collected from sensors. Similarly, in RRP-ECS, each computing resource, i.e., CPU, is divided into two partitions by the RRP model as shown in the dotted squares in Fig. 1. Then, we can assign the basic control tasks of the device, e.g., monitoring temperatures, to the **reserved** partition. For instance, in Fig. 1, Partitions 1, 3 up to $n-1$ are reserved partitions that work on basic control tasks independently. We refer to other partitions as **redundant** partitions as they are not necessary for basic controls. Hence, we can utilize the redundant partitions across different devices, just like Partitions 2, 4 up to n in Fig. 1, to build a virtual machine (VM). Each partition has a VCPU pinned on it while Host OS runs the Application Level on these VCPUs. High-level tasks of smart devices are uploaded to VM where they are accordingly allocated to partitions. Besides, with a host OS installed in VM, the VM can also be connected through a personal computer and handle tasks uploaded by users.

†This paper was supported in part by the US National Science Foundation under Award No. 1219082.

There are several advantages of adopting RRP-ECS.

1) **Efficiency**: With an efficient algorithm dividing resources into partitions [1] at the resource level, redundant computing resources in smart devices can be utilized efficiently to assist other smart devices instead of being idle.
2) **Security**: RRP guarantees that reserved partitions can work independently without interrupts from other partitions albeit they are located on the same physical core and thus the security of basic control tasks is guaranteed.
3) **Portability**: Different smart objects always have heterogeneity in either hardware or software. With the centralized structure introduced later, the addition and removal of CPUs in RRP-ECS are less costly. Similarly, other resources, e.g., hard disk, can easily be included or removed with a centralized manager.
4) **Transparency**: With all tasks managed by the VM, if the local computing resource is not enough, VM can directly get connected to the cloud. Thus the cloud connection and resource usage are transparent to Applications.

The remainder of this paper discusses: 1) The structure of RRP-ECS. 2) The resource interface provided by the RRP model. 3) Potential strategies that map tasks to partitions and their performances.

II. The structure of RRP-ECS

As shown in Fig. 1, Virtual machine consists of different partitions in the resource level. Then a host OS is run on the resource level where applications are deployed. In this section, we show how these are done based on the centralized structure of RRP-ECS. Reserved partitions work independently and are not included in the VM and there is **only one VM** which employs the centralized structure as shown in Fig. 2.

In the centralized structure, the Task Manager (TM) is responsible for allocating tasks to partitions and is similar to the Domain 0 of Xen [2] in implementation. TM may reside in an independent CPU for safety. However, it can also be a partition relying on its degree of load, which will be discussed in the future. The function of TM includes: managing the tasks coming from other partitions, maintaining device drivers and communicating hardware information to partitions. TM starts first during the booting of the system and then starts managing redundant partitions. In a nutshell, TM acts as a nerve center for the VM. Due to limitations of space, we mainly focus on the task processing, which is the main function of TM.

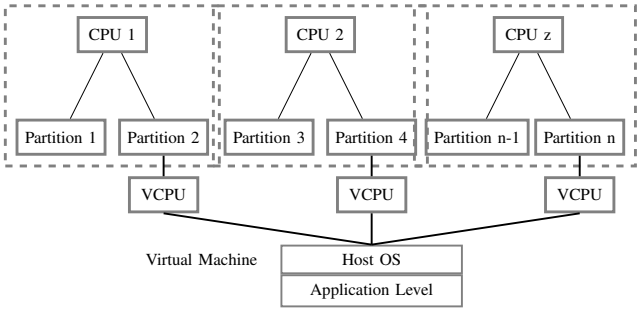


Fig. 1: A two-level resource partitioning framework.

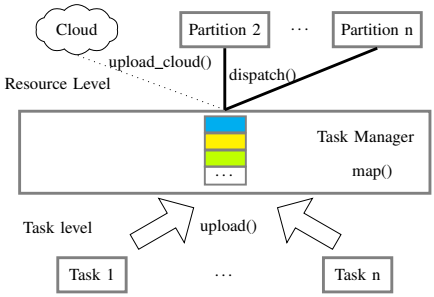


Fig. 2: Centralized Task-Partition Architecture.

Tasks from different Applications are all managed by TM. To be specific, there are two sources of tasks. 1) Smart devices: High-level intensive tasks from smart devices are uploaded to TM. 2) Host OS: The host OS installed in the VM can be reached through Secure Shell (SSH) like cloud. Tasks uploaded through the connection are also uploaded to TM.

TM processes the tasks with the following operations, which is also shown in Fig. 2.

1) **upload()**: Tasks from different sources are uploaded to TM through upload().
2) **map()**: After being uploaded, tasks are put into a global queue, where the LLF (least laxity first) policy [3] is adopted. Tasks popped out from the queue will be mapped to a suitable partition for execution if there exists one.
3) **dispatch()**: If a task is successfully mapped to a partition, it will be transmitted to the partition through event channels and executed there. Event channels are built based on TCP/IP [2].
4) **upload_cloud()**: If no partitions can accomplish the task on time, it will be uploaded to the cloud and taken good care of there [4].

The advantages of adopting a centralized structure in RRP-ECS are as follows. 1) With a centralized TM, which acts like Domain 0 in Xen, merely installing a drive on CPUs in smart devices can include them in RRP-ECS. This makes the removal and addition of devices easier. 2) A centralized structure makes the connection to the resource level, i.e., clouds and partitions, transparent to Applications. 3) A centralized structure reduces the communications between partitions, which may lead the system to potential intrusions or vulnerability.

With a centralized TM, the biggest challenge for RRP-ECS is developing policy mapping tasks to partitions. Without a proper policy, tasks may be assigned to a partition where it cannot be completed on time. Therefore, a schedulability test on each partition and a proper mapping policy are much required. To better understand this, we introduce the resource interface provided by RRP.

III. Resource interface provided by the RRP Model

We first formally define a **time slice**.

Definition 1: Time slice is the basic unit of time containing fixed number of cycles that cannot be further partitioned in any given resource partition.

Based on time slices, RRP divides computing resources into partitions efficiently based on ***availability factors*** and ***supply regularities***. Availability factor $\alpha(A)$ shows how many time slices a certain partition A takes from the CPU during the hyper-period. Supply regularity $k(A)$ shows how close A is to an average distribution [1]. In this paper, we are only going to discuss regular partition whose supply regularity is 1.

Regular partitions are punctual as shown in Theorem 1.

Theorem 1: A regular partition A will have exactly one time slice every $\frac{1}{\alpha(A)}$ time slices.

The proof of Theorem 1 is given in the technical report [5].

The resource interface of a regular partition is simply the **availability factor α**. Thanks to the definition of time slice, the availability factor can be applied to evaluate the computing ability of partitions located on uniform processors.

IV. Task Mapping Problem

Since [6] already proves that scheduling a task set on a partition is the same as scheduling a task set on a CPU, we will not discuss the scheduling of tasks on a single partition here. With the structure in Section II and the resource interface provided by the RRP illustrated in Section III, the main challenge of RRP-ECS is to make sure the mapping policy in map() neither makes partitions overloaded nor waste enormous resources. For that, we first introduce a schedulability test to judge whether a task is schedulable on a partition.

A. Schedulability test for task mapping problem in RRP-ECS

We first define the task set: $T = \{T_1, T_2, ..., T_n\}$. Each task T_i is defined by $T_i = (c_i, d_i, p_i, R_i, D_i, V_i)$. c_i, d_i and p_i respectively refer to the Worst Case Execution Time (WCET), the relative deadline and the period of the task. V_i is the value of T_i, which is the reward for accomplishing T_i on time. R_i is the arrival time of the task. No information of T_i can be accessed before R_i.

Similarly, D_i is the departure time of task T_i. A periodic task T_i does not leave the system, i.e., $D_i = +\infty$. Sporadic tasks, on the other hand, are split by TM into multiple single-instance sporadic tasks. Hence, a single-instance sporadic task T_k has a departure time $D_k = R_k + min(d_k, p_k)$ since it is supposed to leave once its execution is done.

The regular partition set is $\mathbb{P} = \{\mathbb{P}_1, \mathbb{P}_2, ..., \mathbb{P}_m\}$. Each partition $\mathbb{P}_j$ is defined by the resource interface $\mathbb{P}_j = (\alpha_j)$ where α_j is the availability factor of $\mathbb{P}_j$.

We give the schedulability test for tasks on a regular partition.

Theorem 2: A task set T is schedulable on a regular partition $\mathbb{P}_j$ if

$$\sum_{i=1}^{|T|} \frac{c_i}{\min\{d_i, p_i\}} \leq \alpha_j \tag{1}$$

The proof of Theorem 2 is included in the technical report [5]. It is worth mentioning that, for implicit task sets, Equation 1 is the necessary and sufficient condition.

With Theorem 2, we can further define the notation of density.

Definition 2: The density of a task T_i is

$$DEN_i = \frac{c_i}{\min\{d_i, p_i\}} \tag{2}$$

B. Model Formulation

Now we formulate the mapping problem. We first consider the model consists of only periodic tasks involved. The constraints are as discussed below.

- ***Schedulability constraint***: The sum of the densities of tasks allocated to a certain partition P_i should not exceed α_i to keep tasks real-time according to Theorem 2.
- ***Fair constraint***: A task will be abandoned only when no partition can accommodate it because RRP-ECS is not allowed to abandon tasks freely.
- ***Online constraint***: No information regarding task T_i can be known before its arrival time R_i.
- ***Non-migration constraint***: Once a task is assigned to a partition, it cannot be migrated to another partition or removed from the system until its departure time.

With the above constraints, the ultimate goal is to maximize the expression:

$$\max \sum_{i=1}^{n} x_i V_i \tag{3}$$

In equation 3, x_i is a binary variable showing whether T_i is accomplished on time. If T_i is accomplished on time, $x_i = 1$, else $x_i = 0$. When $x_i = 0$, T_i will be uploaded to the cloud and executed there because the resource in RRP-ECS cannot fulfill its request on time.

We name this model Regular-partitions Periodic tasks Mapping (RPM). Similarly, we can formulate Regular-partitions Composite tasks Mapping (RCM) consists of both periodic and sporadic tasks.

For RCM, all constraints and the goal above remain the same while a new constraint is added.

- ***Leaving constraint***: No resources will be reserved for task T_i after D_i.

The goal of RCM is also to maximize the values gained as shown in Equation 3.

For both RPM and RCM, we mainly consider two sorts of value of tasks. 1) Unit case: All tasks are equally important, i.e., $V_i = 1, \forall i \in [1, n]$. 2) General case: The value of each task is decided by the user arbitrarily. Here we do not discuss the proportional case where $V_i = DEN_i$ due to space limitations, but it is discussed in the technical report [5].

C. Compare and contrast

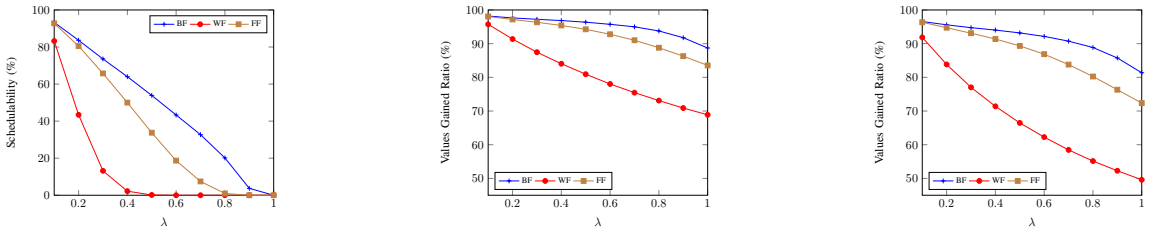
Though RRP has been extensively studied since its proposal [1, 6], mapping tasks to partitions has never been discussed because this is the first time RRP is used to integrate distributed resources into one VM.

Despite that RPM and RCM resemble task mapping on uniform processors and Multiple Knapsack Problem (MKP), some characteristics distinguish RPM and RCM from models in former works. 1) Both RPM and RCM are online models, while many decent works are done in offline scenarios [7, 8]. The lack of preliminary information and long response time requirements prevent offline algorithms from solving the online version. 2) Both densities of tasks and the availability factors of partitions do not have basic unit size, thus breaking the assumption that the capacity of bins and size of items are integers [9]. 3) Unlike in removable cases [10], the fair constraint does not allow the system to abandon tasks freely.

Therefore, RPM and RCM are different from former models. The solutions applied to solve them, however, decides the efficiency of RRP-ECS. Next, we give our observations.

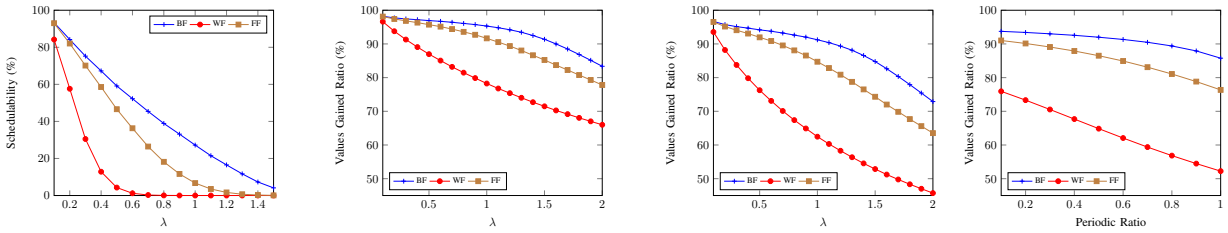
D. Algorithm Best-Fit

Our answer towards RPM and RCM is quite simple: Best Fit (BF). In BF, we use a set of parameter $\{\beta_j\}$ to record the capacity of corresponding partitions $\{\mathbb{P}_j\}$. Initially, we set $\beta_j = \alpha_j$. For task T_i, BF chooses partition P_k satisfying



(a) Schedulability (b) Values Gained Ratio in Unit Case (c) Values Gained Ratio in General Case

Fig. 3: Performance of various algorithms with variable ratios (λ) between total utilization of tasks and total availability factors of partitions in RPM.



(a) Schedulability (b) Values Gained Ratio in Unit Case (c) Values Gained Ratio in General Case (d) Influence of Periodic Ratio when $\lambda = 0.9$ in General Case

Fig. 4: Performance of various algorithms with variable ratios (λ) between total utilization of tasks and total availability factors of partitions in RCM and the influence of Periodic Ratio in RCM.

Equation 4. The key idea of BF is to reserve partitions with large capacities for the future.

$$k = \min_{j:\beta_j \geq DEN_i} \beta_j \tag{4}$$

Though maximization problem in RPM and RCM are both NP-hard problem in strong sense as variants of MKP [8], we prove a constant competitive ratio for BF on accommodating inputs [11].

Theorem 3: Let ω be an accommodating sequence [11]. $BF(\omega)$, the result of BF on ω and $OPT(\omega)$, the result of the offline optimal algorithm on ω satisfies: $\frac{BF(\omega)}{OPT(\omega)} \geq \frac{1}{2}$
The proof is illustrated in the technical report [5].

Intuitively speaking, BF performs well because it always reserves the resource for the unknown future. It may not be a good strategy in offline scenarios [8], but BF performs really well by being prudent in online scenarios. Next we validate the performance of BF through simulations [1].

To make the simulation more realistic, we only pass in the sum of the availability factors of partitions and the sum of densities of tasks. The number of partitions and tasks as well as the specific parameters of each partition and task are all randomly generated. Each simulation is repeated $1,000,000$ times and the results shown are the average value of results. Best Fit is compared with Worst Fit (WF) and First Fit (FF), which are prominent online algorithms discussed before [12].

We measure the result by **Schedulability**, which reveals the percentage of the task sets that are schedulable and **Values Gained Ratio**, which is the ratio between values gained by the system and the total values of all tasks.

According to Fig. 3 and 4, the performance of BF always overwhelms FF and WF with obvious gaps in all measurements and few tasks fail when BF is applied even for heavy-loaded task sets. Under different Periodic Ratio, which is the ratio between the number of periodic tasks and that of all tasks, BF still performs better throughout as shown in Fig. 4d. The gap between BF and other algorithms increase as the Periodic Ratio increases since a higher periodic ratio implies a heavier load. This heavy load can be attributed to the fact that periodic tasks never leave the system. More simulations are presented in the technical report [5]. In short, all simulations indicate that BF performs well even when the resource is limited and it can guarantee the efficiency of RRP-ECS in most cases.

[1]Simulation codes: https://github.com/DDeChoU/TaskMappingSimulation

V. Conclusion

This paper proposes an edge computing system RRP-ECS and discusses the centralized structure. Based on the resource interface provided by RRP, we further discuss the task mapping problem and provide a brief and an efficient response with Best Fit which is further validated by simulations.

However, we still have several potential to-do tasks which can be pursued in the future regarding RRP-ECS.

- With no theoretical obstacles, the next step is to implement RRP-ECS on Xen.
- The feasibility of using caches (levels L1, 2 & 3) to accelerate the scheduling and provide real-time guarantee in TM needs to be investigated.
- The simulation codes used in this paper provide an environment for the development of Reinforcement Learning (RL) algorithms in RPM and RCM based on Keras [13].

References

[1] Y. Li and A. M. Cheng, "Static approximation algorithms for regularity-based resource partitioning," in *Real-Time Systems Symposium (RTSS), 2012 IEEE 33rd*. IEEE, 2012, pp. 137–148.

[2] D. Chisnall, *The definitive guide to the xen hypervisor*. Pearson Education, 2008.

[3] M. L. Dertouzos and A. K. Mok, "Multiprocessor online scheduling of hard-real-time tasks," *IEEE Transactions on software engineering*, vol. 15, no. 12, pp. 1497–1506, 1989.

[4] A. K. Mishra, J. L. Hellerstein, W. Cirne, and C. R. Das, "Towards characterizing cloud backend workloads: insights from google compute clusters," *ACM SIGMETRICS Performance Evaluation Review*, vol. 37, no. 4, pp. 34–41, 2010.

[5] G. Dai, P. K. Paluri, and A. M. Cheng, "RRP edge computing system," www.dropbox.com/s/awlmdgucmafmaiw/RRPEdgeComputingSystem.pdf?dl=0.

[6] Y. Li and A. M. K. Cheng, "Transparent real-time task scheduling on temporal resource partitions," *IEEE Transactions on Computers*, vol. 65, no. 5, pp. 1646–1655, 2016.

[7] K. Jansen, "A fast approximation scheme for the multiple knapsack problem," in *International Conference on Current Trends in Theory and Practice of Computer Science*. Springer, 2012, pp. 313–324.

[8] C. Guo, X. Hua, H. Wu, D. Lautner, and S. Ren, "Best-harmonically-fit periodic task assignment algorithm on multiple periodic resources." *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 5, pp. 1303–1315, 2016.

[9] A. Borodin, D. Pankratov, and A. Salehi-Abari, "A simple ptas for the dual bin packing problem and advice complexity of its online version," *arXiv preprint arXiv:1708.01657*, 2017.

[10] X. Han, Y. Kawase, and K. Makino, "Randomized algorithms for online knapsack problems," *Theoretical Computer Science*, vol. 562, pp. 395–405, 2015.

[11] J. Boyar, K. S. Larsen, and M. N. Nielsen, "The accommodating function: A generalization of the competitive ratio," *SIAM Journal on Computing*, vol. 31, no. 1, pp. 233–258, 2001.

[12] S. Baruah, "Partitioned edf scheduling: a closer look," *Real-Time Systems*, vol. 49, no. 6, pp. 715–729, 2013.

[13] F. Chollet, "keras," github.com/fchollet/keras, 2015.