Proceedings of the Work-In-Progress Session

21st IEEE Real-Time and Embedded
Technology and Applications Symposium

Edited by Arvind Easwaran
Nanyang Technological University

Seattle, USA
April 14, 2015

Message from the Work-In-Progress Chair

The Work-In-Progress (WIP) session at RTAS is dedicated to new and on-going research in the field of real-time and embedded systems, with emphasis on the application of these technologies to solve practical problems. The WIP session is an integral part of the RTAS program, as it enables researchers to present and discuss early-stage ideas with the wider community and solicit constructive feedback in the process. The WIP schedule comprising brief presentations followed by an extended poster session in a relaxed setting, provides ample opportunities for discussions. Since RTAS is co-located with three other top conferences and several workshops as part of the Cyber-Physical Systems' (CPS) Week, the WIP session provides an ideal setting for interactions with the wider CPS community.

In 2015, the WIP presentation and poster session will be held on the first day of RTAS, starting from 5PM. A total of 16 papers were submitted to the WIP session, out of which 14 papers were selected for the final program. Papers were selected based on an extensive review process by the program committee in which each paper was reviewed by at least three different reviewers. The resulting WIP session has a good mix of theoretical and applied papers, covering a wide range of topics in real-time and embedded systems. I am confident that the program will lead to interesting discussions during the poster session. I would like to thank the authors of all the submitted papers as well as the program committee members for their time and effort.

I invite all of you to join me in taking advantage of this excellent opportunity to learn and interact with our fellow colleagues.

Arvind Easwaran
Nanyang Technological University
Singapore
RTAS 2015 WIP Chair

RTAS 2015 Work-In-Progress Technical Program Committee

Björn Brandenburg, Max Planck Institute for Software Systems, Germany

Cong Liu, University of Texas at Dallas, USA

Eduardo Quiñones, Barcelona Supercomputing Center, Spain

Heechul Yun, University of Kansas, USA

Jinkyu Lee, Sungkyunkwan University, Republic of Korea

Linh Thi Xuan Phan, University of Pennsylvania, USA

Moris Behnam, Mälardalen University, Sweden

Praveen Jayachandran, IBM Research, India

Rodolfo Pellizoni, University of Waterloo, Canada

Technical Program

Real-Time Logic (RTL) Modeling of SpaceWire-D Networks

Qiang Zhou, Zikun Yang
School of Electronics Information Eng.
BeiHang University
Beijing, 100191, China
Email: zhouqiang_ee@buaa.edu.cn, yzk920205@163.com

Albert M. K. Cheng
Dept. of Computer Science
University of Houston
Houston, TX 77004, United States
Email: cheng@cs.uh.edu

***Abstract*—SpaceWire is a standard for on-board satellite networks as the basis for future data-handling architectures. However, it cannot meet the deterministic requirement for safety/time critical applications in spacecraft. Therefore, SpaceWire-D is developed to provide deterministic delivery over a SpaceWire network. Formal analysis and verification of real-time systems is critical to their development and safe implementation, and is a prerequisite for obtaining their safety certification. Failure to meet specified timing constraints such as deadlines in hard real-time systems may lead to catastrophic results. In this paper, Real-Time Logic (RTL) is used to specify and verify timing properties of the SpaceWire-D network. Based on the principles of the SpaceWire-D protocol, we first analyze the timing properties of the fundamental transaction RMAP WRITE. After that, its structure is modeled in RTL and Presburger Arithmetic representations. Then, the associated constraint graph and safety analysis is provided. Finally, it is suggested that the RTL-based verification method can be useful for protocol evaluation and provision of recommendation for further protocol evolutions.**

Keywords*—SpaceWire-D, real-time logic (RTL), Time-code, Formal verification.

I. Introduction

SpaceWire is a standard for on-board satellite networks chosen by the ESA as the basis for future data-handling architectures [1]. It delivers the high throughput required for payload data with low implementation cost. However, it does not provide guarantees in the packet latency due to network congestion. To guarantee the on-time transmission of real-time message streams, SpaceWire-D is developed to provide deterministic delivery over a SpaceWire network [2].

Real-time systems can be defined by either a structural specification or a behavioral specification. A behavioral specification often suffices for verifying the timing properties of the system [3]. Formal analysis and verification of real-time systems are required for their development and safe implementation, and is a prerequisite for obtaining their safety certification. Failure to meet specified timing constraints such as deadlines in hard real-time systems may lead to catastrophic results [4]. In order to specify and verify timing properties of a SpaceWire-D network, we propose using Real-Time Logic (RTL), which can describe the specification and the desired safety assertion. Based on the principles of the SpaceWire-D protocol, we first analyze the timing properties of the fundamental transaction Remote Memory Access Protocol (RMAP) WRITE. After that, its structure is modeled in RTL and Presburger Arithmetic representations. Then, the associated constraint graph and safety analysis is provided.

This work is sponsored in part by the State Scholarship Fund of China under award No. 201303070189, by the Fundamental Research Funds for the Central Universities, by the Beijing Natural Science Foundation under award No. 4133089, and by the US National Science Foundation under award Nos. 0720856 and 1219082.

II. Background and Related Works

A. Real-Time Logic Formal Method

Real-Time Logic (RTL) is a multi-sorted first-order logic defined by Jahanian and Mok [5]. RTL supports reasoning about occurrences of events and actions. An action is an activity that requires a non-zero amount of system resources. Start/end events (denoted by $\uparrow A$ and $\downarrow A$) mark the initiation and completion of an action A. The time of occurrence of the i-th instance of an event E is denoted by the occurrence function $@(E, i)$. With this notation, timing constraints can be expressed as restrictions on the occurrence function. A set of co-operating tasks with precedence constraints can be viewed as an acyclic, directed graph with weights on the edges. The maximum response time specifies the upper bound of the execution time of a precedence graph, i.e., the time interval between the start of the first stimulus task till the completion of the last response task. The precedence relation between two tasks (allocated to the same component) can be expressed by the RTL formula of equation 1.

$$\forall_i @(\downarrow Task\_A, i) < @(\uparrow Task\_B, i) \quad (1)$$

This formula defines that for all instances of $Task\_A$ and $Task\_B$, the start event of task $Task\_B$ has to occur after the occurrence of the end event of the preceding task $Task\_A$.

In RTL, a set of system specifications (SP) and safety assertions (SA) are given, respectively. In order to prove that SA is derivable from SP, the proof by contradiction method is applied, i.e., $SP \rightarrow SA$ is shown through the negated logically equivalent form $SP \wedge \neg SA$, where $\neg SA$ is the negated version of SA. Afterward, a constraint graph is built. Finally, the unsatisfiability of $SP \wedge \neg SA$ is proven if positive cycles are found in the constraint graph [5].

B. SpaceWire-D and its Timing Constraints

SpaceWire-D is a protocol that provides deterministic delivery over a SpaceWire network [2]. SpaceWire-D delivers data within predetermined time constraints.

1) RMAP Transactions: Information shall be passed between SpaceWire nodes using the RMAP protocol. This permits the reading and writing of registers and memory in a remote target node over a SpaceWire network. These operations are considered sufficient for payload data-handling and command and control applications. There are some constraints on the RMAP implementation, e.g., on the time taken to respond to an RMAP command, and on the maximum amount of information that can be read or written by a single RMAP command (see section 3.8 in [2]).

2) Constraints of RMAP Transaction: In a transaction, there are two components, an initiator and target. To guarantee the completion of the transaction, both need to satisfy some implementation constraints. For simplicity without losing the generality, we here focus on the operation of an initiator and target performing a WRITE transaction during a time-slot [4].

3) Sub-tasks and their Timing Constraints: The whole WRITE transaction can be divided into nine sub-tasks, see Table I. To guarantee the completion of the whole WRITE transaction before its deadline, the time which take to finish it must be no more than one Time-Code. In the following RTL analysis, this constraint can also be called as safety assertion.

TABLE I. SUB-TASKS AND THEIR TIMING CONSTRAINTS

	Sub-task (event)	Constraints	Ref.[4]
a	The time taken from the receipt of a time-code to the starting to send an RMAP command from an initiator	T_a	3.8.1(c)
b	Network propagation delay	T_b	
c	RMAP command received at target	T_c	
d	Authorization delay	T_d	3.8.2(a)
e	Data written to memory in target	T_e	3.8.1(a)
f	Additional latency writing data	T_f	3.8.2(b)
g	RMAP reply send delay	T_g	3.8.2(c)
h	Network propagation delay for reply	T_h	
i	RMAP reply received at initiator	T_i	

III. OUR WORK

A. Timing Analysis Using RTL

To analyze and verify system timing properties, we use RTL to specify the safety-critical aspects of the transaction WRITE in SpaceWire-D networks. This representation allows timing and safety analysis, and may be extensible to represent broader aspects of the SpaceWire-D networks. The WRITE transaction and the associated safety assertion are modeled. After converting the RTL representation into a Presburger Arithmetic format and ultimately a constraint graph, cycle analysis verifies safety assertion satisfaction. Here, the scenario parameters are defined as in Table II.

1) RTL Representation: i.workloads for each event: $\forall_i @(\uparrow Task\_A, i) + T_a \geq @(\downarrow Task\_A, i), \cdots$ (omitted due to space limits); ii.precedence: (between start and end events, between end of first task and beginning of next, between beginning of prior task and beginning of next) $\forall_i @(\uparrow Task\_A, i) \leq @(\downarrow Task\_A, i), \cdots$ (omitted due to space limits); iii.Safety Assertion:$\forall_i @(\downarrow Task\_I, i) \leq @(\uparrow Task\_A, i) + 100us$; iv.Negation of Safety Assertion in RTL:$\exists_i @(\uparrow Task\_A, i) + 101us \leq @(\downarrow Task\_I, i)$.

2) RTL Representation Converted to Presburger Arithmetic: The RTL formulas are converted to the Presburger arithmetic into aid in subsequent graphing. The notation conversion is to use a "$S\_$" or an "$E\_$" to represent the task start or end events, respectively. For example, $E\_Task\_A(i) - T_a \leq S\_Task\_A(i), \cdots$ (omitted due to space limits).

3) Constraint Graph Analysis: In order to verify the satisfaction of the safety assertion, the above-mentioned formulas are represented in the constraint graph shown in Fig.1. The system specification alone produces a graph with no positive cycles (without considering the dashed line). Negation of the safety assertion, however, as shown in the dashed line, when graphed, yields edges which produce cycles with positive weights (the sum of the weights on the edges is positive), thus verifies critical system performance.

TABLE II. THE SCENARIO PARAMETERS

Parameters	Value
Date rate	r=200Mbps
No. of the routers	R=4
Time-Code	100μs

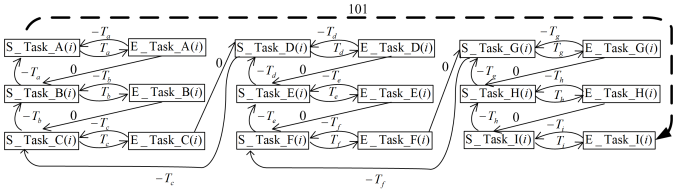


Fig. 1. Constraint Graph

IV. FUTURE WORK

The SpaceWire-D networks system development is actively progressing through requirement and design phases. It is suggested that the RTL representation can be a promising mechanism for similar timing analysis and verification. The RTL representation presented here represents one aspect, the WRITE transaction timing, of a SpaceWire-D network. The specification language itself may be well-suited for representing this as well as possibly a broader range of areas of the system specification. For example, it may be used in the analysis and verification of the performance of the periodic Time-Code synchronization. Also, RTL-based verification method can be useful for protocol evaluation and provision of recommendation for further protocol evolutions.

REFERENCES

[1] ECSS, "SpaceWire - Links, nodes, routers and networks ", *ECSS-E-ST-50-12C*, July 2008, available from http://www.ecss.nl.

[2] S. Parkes and A. Ferrer-Florit, "SpaceWire-D Deterministic Control and Data Delivery Over SpaceWire Networks, "*ESA Contract No. 220774-07-NL/LvH, University of Dundee*, April 2010.

[3] G. Fohler, C. Huber, "Integration of RTL and precedence graphs with a static scheduler as verifier, "*Real-Time Systems, 1994. Proceedings., Sixth Euromicro Workshop on. IEEE*, pp.22-25,1994.

[4] A. M. K. Cheng, H. Niktab, M. Walston, "Timing Analysis of Small Aircraft Transportation System (SATS), "*Embedded and Real-Time Computing Systems and Applications (RTCSA), 2012 IEEE 18th International Conference on*, pp.58-67, 2012.

[5] F. Jahanian, A. K. Mok, "A Graph-Theoretic Approach for Timing Analysis in Real Time Logic, "*RTSS*,pp. 98-108,1986.

Distributing the Complexity of Schedulability Tests

Vladimir Nikolov, Kilian Kempf
Inst. of Inform. Resource Management
Ulm University
D-89069 Ulm, Germany

Franz J. Hauck
Inst. of Distributed Systems
Ulm University
D-89069 Ulm, Germany

Dieter Rautenbach
Inst. of Optimization and OR
Ulm University
D-89069 Ulm, Germany

***Abstract*—Soft real-time systems that have to deal with unknown, dynamically loaded and stopped applications are hard to analyse for feasibility. We present an adaptive hierarchical scheduling model for a dynamic set of applications with fluctuating resource demands. On the lowest level our scheduler ensures correct execution of application tasks according to online approximations of their demands. On the higher levels isolation between the apps is provided while the scheduler continuously re-calibrates given capacities and performs local schedulability analysis for application tasks. In bottleneck situations an optimal degradation is applied by assigning one of different quality levels to the apps. In our architecture the problem of proving schedulability is spatially partitioned and temporally distributed. Tests are performed portion-wise only when and where necessary. Our model maintains feasibility until cost estimations violate custom safety buffers. In such cases a new optimal resource configuration is computed changing the quality of some apps. Periodic violations are also dedected suppressing recurring reconfigurations.**

I. Introduction

Multiple application modes for optimal quality and overload management are not a novelty. As we have shown in [6], [5] most solutions lack online cost refinements and feasibility analysis to handle fluctuations. Others suffer from continuous mode-switches and instability. The exploding solution space further exacerbates an exact schedulability test. This is impractical for real application scenarios, and therefore we propose a new scheduling model that copes with these limitations.

II. System Model

We assume a dynamic set of n concurrent applications $A_1,\ A_2, ..., A_n$, and for each A_i there is a set of modes $M_{i,1}, M_{i,2}, ..., M_{i,m_i} \in \mathcal{M}_i$, which can be switched dynamically. Each application further consists of a set of k tasks $\tau_{i,k} \in \mathcal{T}_i$ also having different modes, e.g. with different periods $T_{i,k}$, deadlines $D_{i,k}$ and costs $C_{i,k}$. Thus, an application mode manifests as a configuration of task modes. In general, a higher application mode has a higher computational demand $R_{i,j}$ but delivers a better quality, e.g. a better video resolution. The quality is formalized by application-specific utility functions $\mathrm{u}_i(M_{i,j})$ given by developers. Each application A_i contains a mode $M_{i,0}$ with $R_{i,0} = \mathrm{u}_i(M_{i,0}) = 0$ which corresponds to the application being temporarily stopped. Adding such a mode ensures the presence of an optimal resource distribution. Applications can be weighted by users or the system itself with importance factors a_i. These factors affect most obviously each applications prominence during the resource distribution process. For further discussion on conceptual and technical mechanisms for the realization of that model, we refer to [6].

This research has been supported by DFG under Grant No. HA2207/8-1 (ARTOS), BMBF under Grant No. 01/H13003 (MyThOS) and EU FP7-ICT-2013.3.4 Project ref: 610686 (POLCA)

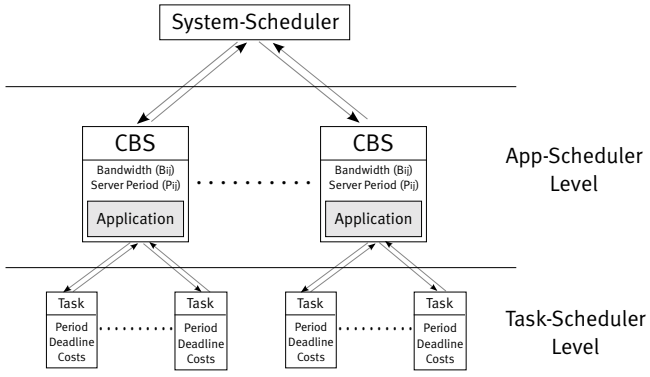


Fig. 1: Scheduling Model

Obviously, a straight-forward approach would have to perform $|\mathcal{M}_1| \times |\mathcal{M}_2| \times \ldots \times |\mathcal{M}_n|$ feasibility tests for all application mode combinations in order to (a) select an optimal modes configuration and (b) ensure schedulability of all application tasks. Assuming a system working with online monitored task costs, when and how often should these tests be performed?

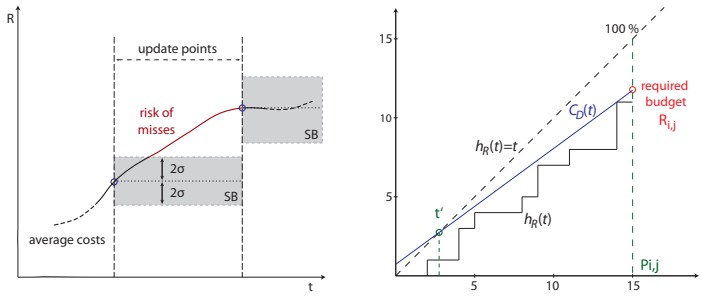
III. Hierarchical Scheduling Model

We propose a three-stage hierarchical scheduling model, as depicted in Figure 1, in order to (a) distribute schedulability checks over time, (b) perform them only where and when needed and (c) optimize the selection of active modes.

An app-local *Task-Scheduler* activates tasks by EDF, monitors their costs and gathers statistical information. Sampled task costs per period are smoothed (SES), and their double standard deviation[1] (2σ) is used as a mode-specific jitter-dependent safety buffer (SB). SBs are set up on task mode activations with the latest values. They are updated only if the measured task costs exceed them (see Fig. 2a). Although task costs cannot be expected to be normally distributed, the 2σ SBs help to detect deviations from a steady approximated state – i.e. outliers are interpreted as bursts. Moreover, FFT and autocorrelation are applied on the samples in order to detect burst cycles. The values are then fed to a higher-level *Application-Scheduler*.

Each application is encapsulated and isolated by a separate *Constant Bandwidth Server* (CBS) [1]. The CBS are configured with period = hyperperiod $P_{i,j}$ of the served app-tasks for their current modes. At the begin of each period a server bandwidth $B_{i,j} = R_{i,j}/P_{i,j}$ may be updated, only if any served task has left its predefined SB (Fig. 2a). If so, the cost budget $R_{i,j}$ needed to successfully schedule the served task set $\mathcal{T}_i$ within $P_{i,j}$ is computed. Thus, bandwidth calculation and feasibility analysis are performed in one step with the latest processor

[1] covering 95,5 % of the sampled values



(a) Safety Buffer Update (b) Capacity Computation Example

Fig. 2: Scheduler Adaptation

demand estimations. For simplicity we assume a synchronous tasks start, and $U_i = \sum_k C_{i,k}/T_{i,k}$ is the partial utilization of task set $\mathcal{T}_i$. The required budget $R_{i,j}$ is determined by the intersection of the tangent $C_D(t) = t \cdot U_i + U_i \cdot max(T_{i,k} - D_{i,k})$ approximating [3] tasks demand bound function $h_R(t)$ and the hyperperiod $P_{i,j}$. Thus, $R_{i,j} = C_D(P_{i,j})$ and according to [3] the feasibility test $h_R(t) \leq t$ must be preformed only for deadline points up to $t' = U_i/(1 - U_i) \cdot max(T_{i,k} - D_{i,k})$. Fig. 2b gives an example for two tasks $\tau_{i,k} = (T, C, D)$ with $\tau_{i,1} = (5, 2, 4)$ and $\tau_{i,2} = (3, 1, 2)$, req. budget $R_{i,j} = 11, 73$, $t' = 2, 75$ and $B_{i,j} = 0, 78$. Currently, we further optimize the feasibility test by applying the superposition approach in [2].

An *Application-Scheduler* schedules the CBS according to their deadlines ($D_{i,j} = P_{i,j}$) by EDF. CBSs keep bottlenecks localized to their origins per definition. We extended our model with a capacity sharing mechanism (CASH) [4]. It reallocates spare capacities to current hot spots and thus delays coarse-grained reconfigurations until total resource exhaustion.

In case of a permanent overload condition, i.e. $\sum_{i=1}^{n} B_{i,j} > 1$, the *System-Scheduler* has to compute a new application-modes configuration with the current estimated bandwidths. Thereby, apps may be even temporarily deactivated. This is described as an optimization problem which aims to maximize the gained utility while the total resource boundary R is preserved:

$$\begin{aligned} \text{Find a selection} \quad J = \; & \{M_{1,j_1}, M_{2,j_2}, ..., M_{n,j_n}\}, \\ \text{maximizing} \quad & \textstyle\sum_{i=1}^{n} a_i \cdot \mathrm{u}_i(M_{i,j_i}), \\ \text{subject to} \quad & \textstyle\sum_{i=1}^{n} B_{i,j_i} \leq R. \end{aligned}$$

IV. Optimization Algorithm

The optimization problem is solved online with a knapsack algorithm and dynamic programming. First, $B_{i,j}$ are cast to integer percentages of R. Two tables, d and J, are used for the computation. For $i \in \{1, 2, ..., n\}$, $r \in \{0, 1, ..., R\}$ and $M_{i,j} \in \mathcal{M}_i$ let

$$\mathrm{d}(i, r) = \textstyle\sum_i a_i \cdot \mathrm{u}_i(M_{i,j_i}) \qquad (1)$$

denote the maximum utility such that $\sum_i B_{i,j_i} \leq r$. Let further

$$\mathrm{J}(i, r) = \{M_{1,j_1}, M_{2,j_2}, ..., M_{i,j_i}\} \qquad (2)$$

be an optimal selection satisfying the given constraints. Table $\mathrm{d}(i, r)$ vividly contains the maximum possible utility for the first i applications and a resource boundary of r, as $\mathrm{J}(i, r)$ holds the selection which led to the gained maximum utility.

The tables are initialized with $\mathrm{d}(0, r) = 0$ and $\mathrm{J}(0, r) = \emptyset$ for all $r = 0, 1, ..., R$. For $i \in \{1, 2, ..., n\}$ and $r \in \{0, 1, ..., R\}$ the following recursion applies:

$$\mathrm{d}(i, r) = \max_{M_{i,j} \in \mathcal{M}_i} \{\mathrm{d}(i - 1, r - B_{i,j}) + a_i \cdot \mathrm{u}_i(M_{i,j})\} \qquad (3)$$

$$\text{while with} \quad \mathrm{J}(i, r) = \mathrm{J}(i - 1, r - B_{i,j_i}) \cup \{M_{i,j_i}\} \qquad (4)$$

an optimum realizing selection is given. Checking if $r - B_{i,j}$ still resides within the tables ensures observation of the constraint $\sum_{i=1}^{n} B_{i,j_i} \leq R$. A formulation of the algorithm in pseudo code can be found in [6].

If all involved numerical values are kept within reasonable ranges, the described algorithm runs in pseudo-polynomial time, since the number of computed values in $\mathrm{d}(i, r)$ is $n * (R + 1)$ and for every entry there are at most $m_{max} = \max_{i \in \{1,2,...,n\}}\{|\mathcal{M}_i|\}$ modes. It is obviously beneficial for the computing time to keep the total resource boundary R small. E.g., for $R = 128$, 10 apps with 5 modes each the solution took approx. $170\mu s$ on a dual-core 2x1.4GHz Intel. The increase of complexity for different numbers was evaluated in [6].

V. ARTOS Framework

We developed a soft real-time framework ARTOS based on the proposed model. In our experiments new apps were run with a low priority until their modes were initially estimated. We were able to verify that (a) the system quickly establishes an optimal modes selection respecting priorities (b) it remains feasible when tasks reside within their tolerance limits (c) deadline misses occur only if tasks leave their SBs (Fig. 2a) (d) feasibility analysis is performed only for apps with stronger fluctuations than their buffer adaptation speed (e) SBs and CASH reduce deadline misses occurrence (f) cyclic bursts are detected faster with autocorrelation than with FFT.

In [5] we exemplify suppression of reconfigurations caused by recurring bursts. For this, we use a measured upper bound of tasks burst costs for capacity estimation instead of cost averages. On this way slack is produced which, however, is shared via CASH.

VI. Conclusion

The proposed model does not decrease the complexity required for proving system schedulability, but distributes it over time to spots violating dynamic tolerance limits. Further, quality levels decisions are decoupled from schedulability checks and are performed on abstracted capacity values accumulating tasks demands within defined time intervals. The presented optimization for quality levels selection provides a directed and aware degradation mechanism for the system in bottleneck situations. The proposed model was applied for SLA-conformity and resource management in a cloud platform architecture [5].

References

[1] L. Abeni and G. C. Buttazzo. Integrating Multimedia Applications in Hard Real-Time Systems. In *IEEE Real-Time Systems Symposium*, 1998.

[2] K. Albers and F. Slomka. Efficient feasibility analysis for real-time systems with edf scheduling. In *DATE, 2005. Proceedings*, 2005.

[3] S.K. Baruah, A.K. Mok, and L.E. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Real-Time Systems Symposium, 1990. Proceedings., 11th*, pages 182–190, Dec 1990.

[4] M. Caccamo, G. Buttazzo, and L. Sha. Capacity sharing for overrun control. In *Proc. of the 21st IEEE Real-Time Sys. Symp.—RTSS*, 2000.

[5] V. Nikolov, S. Kachele, F. J. Hauck, and D. Rautenbach. Cloudfarm: An elastic cloud platform with flexible and adaptive resource management. In *Utility and Cloud Computing (UCC), 2014 IEEE/ACM 7th International Conference on*, pages 547–553, Dec 2014.

[6] V. Nikolov, M. Matousek, D. Rautenbach, L. Penso, and F. J. Hauck. ARTOS: System model and optimization algorithm. techn. rep., 2012.

Hybrid Approach to QoS Management in Multicore-based Embedded Databases

Woochul Kang
Embedded Systems Engineering Department
Incheon National University
Email: wchkang@inu.ac.kr

Minhak Lee
Embedded Systems Engineering Department
Incheon National University
Email: mnaklee@gmail.com

***Abstract*—With the ubiquitous deployment of sensors and network connectivity, database capability is required for many embedded systems for systematic management of real-time data. In such embedded systems, supporting the timeliness of tasks accessing databases is an important problem. However, recent multicore-based embedded architectures pose a significant challenge due to potential contention for non-CPU resources between cores. In this work, we propose the hybrid control approach that combines the advantages of the existing feedback-oriented approaches to support the timeliness of data-intensive tasks with less power consumption. Preliminary results, using prototype implementation, show that the proposed approach improves the energy efficiency while still supporting user-specified Quality-of-Service (QoS) goals, such as task timeliness and desired QoD, under unpredictable multicore environments.**

I. Data-Intensive Real-Time Tasks

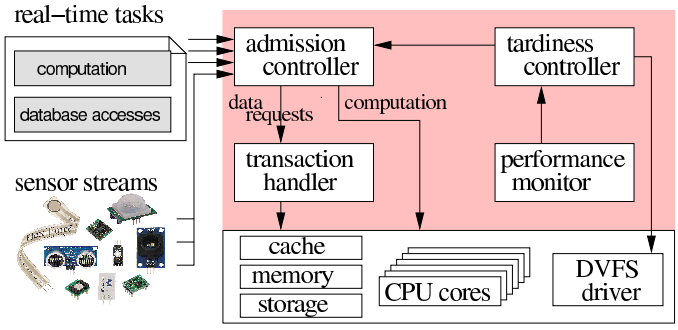


Fig. 1. QoS management architecture for data-intensive real-time tasks.

In traditional real-time systems, tasks are computation-oriented, and most operations are performed on small amount of in-memory data. However, due to ubiquitous deployment of sensors and network connectivity, many real-time operations need to perform on a large amount of real-time data that might be not resident in main memory. For such systems, data from sensors and environments need to be stored and accessed systematically through data services, which are often called as *embedded databases*. As shown in Figure 1, real-time tasks have both computation and data accesses through embedded databases. Such *data-intensive real-time tasks* are invoked either periodically or aperiodically to analyze the current situation using *fresh* sensor data. In multicore platforms, real-time tasks might be executed concurrently in different CPU cores.

In such systems, real-time tasks are required to support a certain level of Quality-of-Service (QoS) in accessing the embedded databases. In this work, for a task J, we define task tardiness, $tard_J$, as a primary QoS metric to measure the timeliness of the task:

$$tard_J = \frac{r_J}{r_J^{target}}, \quad (1)$$

where r_J and r_J^{target} are the actual response time and the target response time of task J, respectively. Another important QoS metric for data-intensive real-time systems is the Quality-of-Data (QoD). In real-time systems, a temporal data object O_i is considered *fresh*, or temporally consistent, if its timestamp is less than its *absolute validity interval (avi)*. To maintain freshness, the update period P_i of temporal data object O_i is set to the half of its absolute validity interval: i.e. $P_i \leq 0.5 \times avi(O_i)$ [1]. In this work, we define QoD as the ratio of fresh data objects to the total number of temporal data objects:

$$QoD = \frac{N_{fresh}}{N_{temporal}}, \quad (2)$$

where N_{fresh} is the total number of fresh data objects and $N_{temporal}$ is the total number of temporal data objects. Since the higher QoD is desirable as far as the system is not overloaded, users or applications specify only the minimum QoD, denoted in QoD_{min}, as a QoS specification.

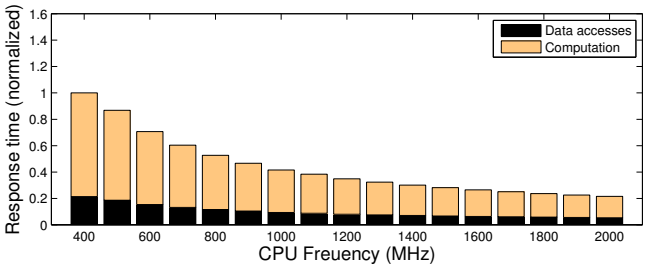
II. QoS Management

In our previous work [2], we presented a reactive QoS management architecture for embedded databases that support QoS using control-theoretic feedback controllers. With the feedback controllers, an embedded database can adjust its control knobs based on current system state to enforce the desired QoS. Figure 1 shows the QoS management architecture. At every sampling period, the *performance monitor* computes the average tardiness of tasks and the *tardiness controller* tunes available control knobs to enforce the desired tardiness.

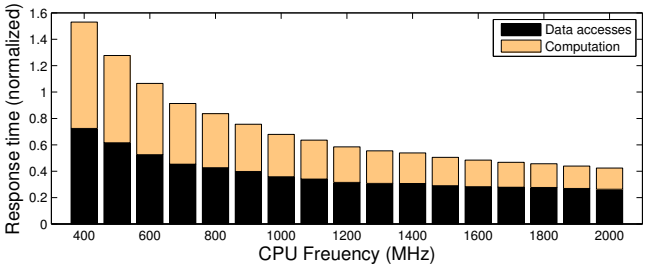
A. Comparison of DVFS and QoD Scaling as Control Knobs

Our previous work exploited Quality-of-Data (QoD) scaling as a major control knob, or actuator, for QoS enforcement. With the QoD scaling, the incoming sensor updates are selectively dropped by the admission controller to control the system workload. However, the range of QoD scaling is limited by users' QoD requirements. Hence, the QoS goals can not be satisfied if QoD is saturated at its maximum or minimum.

In this work, we exploit Dynamic Voltage/Frequency Scaling (DVFS) as a primary control knob for QoS enforcement.



(a) Number of active CPU cores = 1



(b) Number of active CPU cores = 4

Fig. 2. Task response time with varying CPU speeds and CPU cores (QoD=100%).

Unlike QoD scaling, DVFS has a wide range of operating region. For example, our testbed with ARM-based multicore has 19 frequency/voltage levels ranging from 200MHz to 2.0GHz. Further, DVFS is the most successfully deployed power management mechanism since reducing the CPU frequency reduces the dynamic power consumption by a cubic factor.

However, unfortunately, the effectiveness of DVFS is diminished for data-intensive tasks in multicore systems. This is because DVFS mostly affects the speed of computation and can not effectively reduce the contention at non-CPU resources. Figure 2 shows the breakdown of average response time of data-intensive real-time tasks. In this micro-benchmark, 4 concurrent data-intensive tasks are executed periodically in a multicore embedded system. When 4 cores are active (shown in Figure 2-(b)), data accesses take about 360% more time than when only one CPU core is active (shown in Figure 2-(a)). As shown in Figure 2-(b), changing CPU frequency can not effectively control the high ratio of data accessing time to the total response time.

In contrast, when contention between cores for data accesses are high, QoD scaling can be an effective control knob to control the systems performance. Figure 3 shows the result when QoD scaling is applied in the same micro benchmark. As QoD is scaled up/down between 10% to 100%, the data access time is effectively controlled. For example, the response time to access data is decreased by 97% when QoD is changed from 100% to 10%. This result shows that QoD scaling can be an effective method to control the response time of data-intensive tasks, particularly when multicore systems have high contention for non-CPU resources. However, as aforementioned, the range of QoD adaption is determined by user's QoD specification Q_{min}. Therefore, the target QoS can not be satisfied if QoD is saturated at its maximum or minimum.

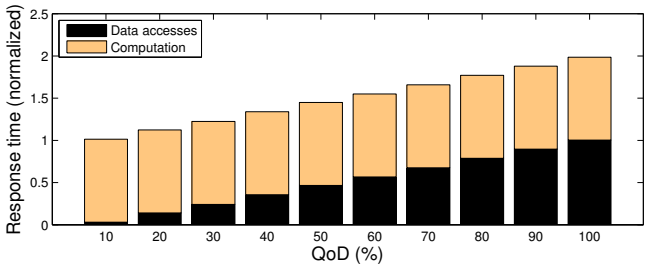


Fig. 3. Task response time with varying QoD (4 cores active at 1.0 GHz).

B. Hybrid Control of QoS

To take advantages of both DVFS and QoD scaling, we propose the *hybrid approach* that integrates DVFS and QoD scaling using hybrid automata. In our approach, DVFS is the major QoS actuator that provides timeliness for data-intensive real-time tasks. However, when significant changes are detected in data access performance, which is common in dynamic multicore systems, the QoD scaling takes over the control to adjust the QoD properly. By scaling down QoD properly, concurrent accesses to databases have less chance of contention for non-CPU resources. Figure 4 shows the hybrid automata with 2 states. At each state, the chosen QoS control mechanism is enforced. For instance, in *DVFS Control* state, the desired tardiness of tasks are supported using DVFS. The guard conditions tell when the state transition should occur.

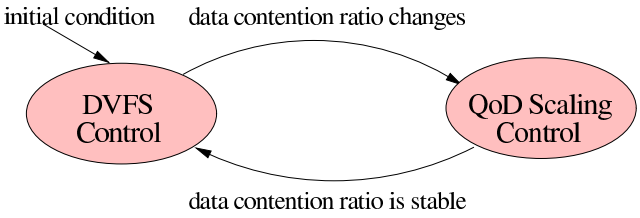


Fig. 4. Switching logic for hybrid control of data-intensive real-time tasks.

III. Preliminary Results

The proposed hybrid control approach is being implemented on an actual multicore-based embedded system. Our preliminary results, using the prototype implementation, show that the hybrid approach can achieve the desired QoS under highly variable multicore environments.

Acknowledgment

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning(2014R1A1A1005781)

References

[1] K.-D. Kang, S. H. Son, and J. A. Stankovic, "Managing deadline miss ratio and sensor data freshness in real-time databases," *IEEE Transacctions on Knowledge and Data Engineering*, vol. 16, no. 10, pp. 1200–1216, October 2004.

[2] W. Kang, S. H. Son, and J. A. Stankovic, "Design, implementation, and evaluation of a qos-aware real-time embedded database," *Computers, IEEE Transactions on*, vol. 61, no. 1, pp. 45–59, 2012.

A Step Toward Deploying Real-Time Applications on Cloud – Modeling Cloud Performance Fluctuation

Hao Wu, Shangping Ren
Illinois Institute of Technology
10 W 31st street, 013
Chicago, IL, USA
{hwu28, ren}@iit.edu

Timm Steven, Gabriele Garzoglio
Fermi National Accelerator Laboratory
Batavia, IL, USA.
{timm, garzogli}@fnal.gov

I. Introduction

Computer clouds are becoming a common platform in many general purpose application domains. However, due to inherent performance uncertainty within computer clouds, applications with real-time and high QoS constraints still operate on traditional computer systems.

As pointed out in [5], one of the main challenges in supporting QoS-critical applications on computer clouds is the computer clouds' performance uncertainty. Phenomena such as VM launching overhead [12], VM performance variation [11] and degradation [10] are documented in the literature and are considered as one of the main obstacles for extending computer cloud's application domain to QoS-critical operation areas.

The run-time differences between computer clouds and traditional computer systems have drawn increased attention from researchers. Cloud modeling tools [9] and different types of cloud performance models [4] are developed. The aims of the established models and modeling tools are to predict application level resource contentions and provide applications with statistical response time guarantees based on queuing theory [4]. However, resource level performance profiling and modeling in a cloud environment is an area that lacks of study, but the need for it is crucial in enabling computer clouds for long-running applications with real-time and QoS constraints.

In this work in progress, we present our ongoing work on modeling cloud performance variations. With the cloud performance model, accurate predictions on task execution times can be obtained hence prediction-based scheduling algorithm can be developed to enable real-time applications also benefit from the elastic computer clouds.

II. Preliminary Result

From the data we have collected from the Fermi National Accelerator Laboratory (Fermilab) private cloud (FermiCloud [2]) and a commercial public cloud (Amazon EC2 [1]), we have observed two important phenomena: (1) virtual machine (VM) launching time (depicted in Fig. 1) and the computing power a VM provides during its operation have

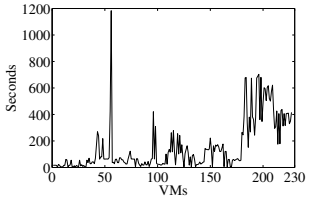


Fig. 1: VM Launching Time on FermiCloud [13]

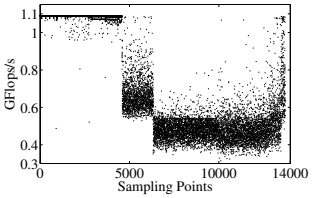


Fig. 2: VM Performance Variation on FermiCloud

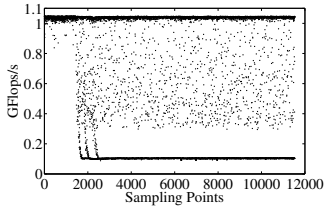


Fig. 3: VM Performance Variation on AWS

large variations irrespective if it is on a private or a public cloud (Fig. 2 and Fig. 3), and (2) VM performance degrades over time even if there is no increase in application workload (Fig. 4).

In particular, Fig. 1 depicts VM launching times measured over 227 VM launches on FermiCloud over a one-month period. Fig. 2 and 3 depict the measurements of VM computing power fluctuation over one month period after a VM is launched on FermiCloud and Amazon EC2, respectively. We refer to VM computing power fluctuation as VM *performance variation*. Fig. 4 shows the execution times of a benchmark application [3] over a three-month period on a FermiCloud VM. The experiment is designed in such a way that potential resource contentions at the VM or the VM's hosting physical machine (PM) levels are removed, i.e., the benchmark application is the only application running on the VM and the VM is the only VM instance hosted by its PM. The benchmark application starts every sixty seconds and terminates once it completes its execution. The application takes less than 260 seconds to complete on a newly launched VM, but takes more than 260 seconds most of the time three months later. We refer to the trend of VM slowing down as VM *performance degradation*.

By reviewing the data we collected on FermiCloud as depicted in Fig. 2 and the corresponding FermiCloud log files, it seems to indicate that the largest VM performance

The research is supported by in part by NSF under grant number CAREER 0746643 and CNS 1018731, by the U.S. Department of Energy under contract number DE-AC02-07CH11359 and by KISTI under a joint Cooperative Research and Development Agreement CRADA-FRA 2014-0002 / KISTI-C14014.

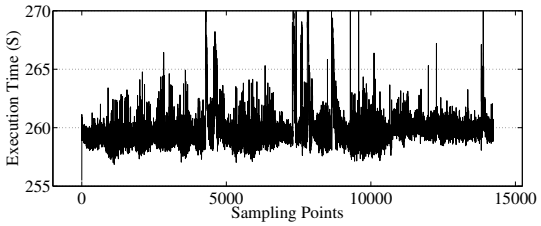


Fig. 4: VM Performance Degradation

change occurs when new VMs are created on the same PM. To understand the correlations between a VM's performance variation and the creation of new VMs on the same PM, we have first studied VM launching procedures and identified the variables that may impact the VM launching process.

A typical launching of a VM has two steps: (1) a VM image is copied from a database to a host PM, and (2) the VM is then booted from the PM. We first identify the factors that may impact both the VM image transferring and VM booting processes. Once these impact factors are identified, we conduct a large set of experiments on pre-production FermiCloud by varying these variables. With over one year data collection, data fitting, and model refining, we have built the VM launching CPU utilization overhead reference models for image transferring ($U_T(t)$) and VM booting($U_B(t)$) [14], [13]:

$$U_T(t) = \frac{1}{1+e^{-0.5(T+t)t}} - \frac{1}{1+e^{-0.5T(t-T)}} \quad (1)$$

$$U_B(t) = \begin{cases} a(t-T) & t < T + \frac{g}{a} \\ \frac{g}{m} e^{-\gamma(t-t_0)(1+IO(t-1))} & otherwise \end{cases} \quad (2)$$

where T is the duration of the image transferring process. It is determined by image file size and the PM's disk write speed. Parameters g, a and γ are system specific constants for peak CPU utilization consumption during the VM booting process, the speed of reaching peak CPU utilization, and utilization decrease rate, respectively, m is the number of CPU cores on the PM, $IO(t-1)$ represents the system's disk write utilization at time $(t-1)$, and $t_0 = T - g/a$.

III. Ongoing Work

In extension to the preliminary reference models we have built for VM launching overhead, we need to further develop a VM performance variation reference model caused by resource contentions after VM creations.

Since a cloud system by its own nature is complex, many variables can cause system performance to change. Researchers have already noticed that VM performance variation is correlated to its workload and its running time [8], [6], [7]. The data we have collected from FermiCloud seems to indicate that a VM's performance is also impacted by the deployment of new VMs on the same PM. Identifying all the factors that may influence a VM's performance and its change is a challenging task, especially for a complex system where possible system configurations (from hardware to software configurations) can be large. Both an engineering investigation and *trial and error* are needed to narrow down the key factors.

On the other hand, VM performance degradation is a slow process. Sometimes, VM performance degradation starts to appear only after VM runs for months. Hence, how to quickly build a reference model for VM performance degradation is a challenging issue. The approach we are to use is to accelerate the VM degradation. We first need to identify factors that can speed up VM degradation. Once we can shorten the data collection interval, similar approaches developed in obtaining VM launching and performance variation models may be applied here by varying experiment settings.

IV. Conclusion

In this paper, we address the problem of cloud performance variation which is one of the main obstacles that prevents real-time applications with high QoS requirements be deployed on cloud. Rather than eliminating the cloud performance variation and degradation which are impractical, if not impossible. We try to study the patterns of cloud performance variation and develop a reference model for the cloud performance variation. With the cloud performance model, accurate predictions on task execution times can be obtained, and hence prediction-based scheduling algorithm can be developed to deploy real-time applications on cloud.

References

[1] Amazon web service. http://aws.amazon.com.

[2] Fermicloud. https://fclweb.fnal.gov.

[3] Pegasus workflow generator. https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator.

[4] D. Ardagna, G. Casale, M. Ciavotta, J. F. Pérez, and W. Wang. Quality-of-service in cloud computing: modeling techniques and their applications. *Journal of Internet Services and Applications*, 5(1):1–17, 2014.

[5] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and Z. Matei. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010.

[6] Y. Bao, X. Sun, and K. Trivedi. A workload-based analysis of software aging, and rejuvenation. *Reliability, IEEE Transactions on*, 54(3):541–548, Sept 2005.

[7] A. Bovenzi, D. Cotroneo, R. Pietrantuono, and S. Russo. Workload characterization for software aging analysis. In *Software Reliability Engineering (ISSRE), 2011 IEEE 22nd International Symposium on*, pages 240–249, Nov 2011.

[8] D. Bruneo, S. Distefano, F. Longo, A. Puliafito, and M. Scarpa. Workload-based software rejuvenation in cloud systems. *Computers, IEEE Transactions on*, 62(6):1072–1085, June 2013.

[9] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, 41(1):23–50, 2011.

[10] D. Cotroneo, R. Natella, R. Pietrantuono, and S. Russo. A survey of software aging and rejuvenation studies. *J. Emerg. Technol. Comput. Syst.*, 10(1):8:1–8:34, Jan. 2014.

[11] J. Dejun, G. Pierre, and C.-H. Chi. Ec2 performance analysis for resource provisioning of service-oriented applications. In *Service-Oriented Computing. ICSOC/ServiceWave 2009 Workshops*, pages 197–207. Springer, 2010.

[12] M. Mao and M. Humphrey. A performance study on the vm startup time in the cloud. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 423–430. IEEE, 2012.

[13] H. Wu, S. Ren, G. Garzoglio, S. Timm, G. Bernabeu, K. Chadwick, and S.-Y. Noh. A reference model for virtual machine launching overhead. *IEEE Transactions on Cloud Computing*, 2014.

[14] H. Wu, S. Ren, G. Garzoglio, S. Timm, G. Bernabeu, and S.-Y. Noh. Modeling the virtual machine launching overhead under fermicloud. In *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on*, pages 374–383. IEEE, 2014.

Probably Right, Probably on Time: An Analysis of CAN in the Presence of Host and Network Faults

Arpan Gujarati*, Akshay Aggarwal†, Allen Clement‡, Björn B. Brandenburg*
*MPI-SWS, Germany †IIT-Kanpur, India ‡Google Inc., Zurich

***Abstract*—Safety-critical real-time systems that operate in electromagnetically noisy environments must be designed to withstand electromagnetic interference. Typically, networked systems handle this issue by retransmitting corrupted messages and by replicating critical processes on independent hosts. However, there exists a fundamental tradeoff between the two techniques: to increase a system's resiliency to message retransmissions without violating any deadlines, a low bus load is favorable; whereas adding replicas requires more messages to be sent, which in turn increases the bus load. This paper presents our ongoing work on a probabilistic analysis that quantifies this tradeoff in CAN-based systems, which enables system designers to select an optimal number of replicas that maximizes the probability of a correct *and* timely operation of a distributed real-time system.**

I. Introduction

Automotive embedded systems are surrounded by spark plugs and electric motors. Industrial embedded systems may be deployed in close vicinity to high-power machinery. Robots may need to operate in environments exposed to hard radiation. All of the above are examples of safety-critical real-time systems that are susceptible to electromagnetic interference (EMI) and must be designed to withstand its effects; including hangs, crashes, or incorrect outputs (*node faults*), and, in networked systems, also corrupted messages (*transmission faults*) [1, 4].

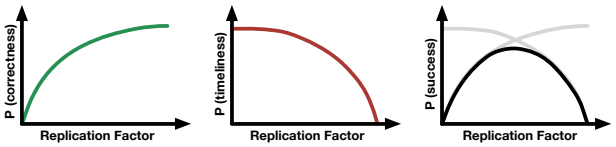
Presently, networked systems tolerate transmission faults by retransmitting corrupted messages (*e.g.*, CAN controllers automatically retransmit a message if any host connected to the bus reports a transmission fault). Node faults on the other hand are typically tolerated by replicating critical processes on independent nodes. In the context of real-time systems, however, both these techniques fundamentally conflict with each other.

While a low bus load is favorable to ensure that deadlines are not violated due to retransmissions (*i.e.*, the more slack, the better), replication increases the bus load as more messages are sent (*i.e.*, it reduces the available slack). Thus, beyond a certain point, adding replicas actually hinders a system's ability to meet all deadlines, and can ultimately decrease its overall reliability.

Our research aims to develop a probabilistic analysis that quantifies the fault-tolerance versus timeliness tradeoff in CAN-based safety-critical real-time systems. The proposed analysis will enable system designers to select an optimal number of replicas that maximizes the probability of a correct *and* timely execution of a distributed real-time system.

II. Key Idea and Challenges

Key idea: We expect the probability of a correct execution to improve with replication due to better tolerance against node faults, but the probability of a timely execution to degrade with replication due to higher bus load, as illustrated in Figs. 1(a) and 1(b). With the proposed analysis, it is possible to determine the probability of a correct *and* timely execution for different replication factors (*i.e.*, number of replicas per tasks), and hence, quantify the benefits of replicating any given task or compute the optimal replication factor (see Fig. 1(c)).



(a) (b) (c)

Fig. 1. *Replication factor* denotes the number of independent replicas of critical tasks. The illustration shows that: **(a)** probability of correct execution improves with replication; **(b)** probability of timely execution degrades with replication; and **(c)** probability of correct *and* timely execution initially increases with replication due to better fault-tolerance, but then starts degrading with replication due to increasing bus load. The graphs are just a conceptual illustration, *i.e.*, they do not represent actual probabilities or empirical results.

Challenges: Although Tindell et al. [5] and Broster et al. [2] have extensively investigated the problem of fault-aware timing analysis for the CAN message bus, a simultaneous analysis of both timeliness and correctness in a system with replicated tasks is significantly more involved. Below, we list some of the major challenges that we have identified in this work.

Assume two tasks A and B running on independent hosts networked via a CAN bus. A regularly sends message m_a to B. Suppose that A is replicated in order to ensure that B's output is always based on a correct input from A, *i.e.*, each replica of A, denoted by A_i, sends a separate message m_a^i to B. Thus, B must implement an aggregation procedure to select/generate the correct input value. In addition, since the hosts may or may not have access to a synchronized clock, and since each message m_a^i may be generated either periodically or sporadically, the aggregation procedure needs to be implemented differently for each scenario (*e.g.*, B cannot know the absolute deadline of each message m_a^i if the clocks are not synchronized).

Another major challenge is to define a new analysis that establishes timeliness guarantees for a set of messages, *e.g.*, how to compute the probability that a "sufficient" number of messages of a larger message set $\{m_a^i\}_i$ are transmitted "on time", where "on time" and "sufficient" may mean that at least a majority of messages reach before B executes its aggregation procedure. Jitter further complicates this exercise. There is also the problem of finding a replication-aware priority assignment that optimizes the safety of highly critical tasks (which are replicated), but without compromising the timeliness guarantees of tasks that are not replicated.

In the next section, we explain an initial probabilistic analysis for hosts with synchronized clocks and periodic messages. In particular, we elaborate on the technique to account for both node faults and transmission faults in the same analysis.

III. Replication-Aware Probabilistic Analysis

System model: We assume a CAN-based system consisting of $r+1$ tasks running on independent hosts, including r replicas of task A and a single instance of task B (as in Sec. II). Each

replica A_i periodically sends message m_a^i to B, with period T_a, deadline D_a, and jitter J_a^i. The j^{th} instance of this message is denoted as $m_{a,j}^i$ and its absolute deadline is denoted as $d_{a,j}$.

Assuming that all the hosts have globally synchronized clocks, B implements the following aggregation protocol for the j^{th} round of messages: at time $d_{a,j}$, B collects all messages in $\{m_{a,j}^i\}_i$ that it has received until $d_{a,j}$, selects the payload that is contained in the majority of these messages, and considers this payload as its input value for the j^{th} round. In this initial work, we assume that B can distinguish between any two messages $m_{a,j}^i$ and $m_{a,k}^i$ transmitted by the same replica.

Fault model: We consider three types of faults: omission faults (failing to send a message), commission faults (sending an incorrect message), and transmission faults (message corruptions on the bus). We assume that messages omitted by a replica A_i do not interfere on the CAN bus, an observation that we exploit in the proposed analysis. We also assume that the host running B executes fault-free. However, we can do away with the latter assumption by replicating B and analyzing the multiple replicas of B individually.

Each fault type is associated with a probabilistic fault model. As assumed by Broster et al. [2], we use a Poisson distribution to model the probability that n transmission faults occur in any time period of length t, *i.e.*, $P_t(n,t)$. For the node faults, we use a simpler model that associates fixed probabilities P_o and P_c with an omission fault and a commission fault per message, respectively. For brevity, we assume the same fault probabilities across all hosts; the full analysis can be extended to incorporate host-specific probabilities. The analysis does not cover "babbling idiot" failures, as we rely on the bus guardian solution proposed earlier by Broster et al. [2].

Analysis: Let $P_{su}(M_j)$ denote the probability of "success", *i.e.*, the probability that for any message set $M_j = \{m_{a,j}^i\}_{1\leq i\leq r}$, the input value inferred by the aggregation protocol at B is indeed correct. Intuitively, $P_{su}(M_j)$ is a combination of three factors: omission faults and commission faults at A, and retransmission delays on the bus. Thus, in order to compute $P_{su}(M_j)$, we use the following approach. **Step 1:** Suppose that $O_j \subseteq M_j$ denotes the set of messages omitted at A. Let the corresponding probability be denoted as $P_{om}(O_j)$. Since messages omitted at A do not interfere on the CAN bus, we analyze only the timeliness and correctness of messages in $O'_j = M \setminus O_j$. **Step 2:** Given O'_j, we then consider $|O'_j|$ cases, where *case* k implies that only k out of $|O'_j|$ messages have response time less than or equal to D_a. Let the corresponding probability of timeliness be denoted as $P_{ti}(O'_j,k)$. **Step 3:** Finally, for each *case* k, *i.e.*, among all the k messages transmitted on time, we consider all combinations of corrupted and correct messages and for each combination decide whether the value inferred by the aggregation protocol at B is correct or wrong. The corresponding probability of correctness is denoted as $P_{co}(O'_j,k)$. Using the aforementioned steps, $P_{su}(M_j)$ can be defined as $P_{su}(M_j) = \sum_{O_j\subseteq M_j} P_{om}(O_j) \cdot \sum_{k\leq|O'_j|} P_{ti}(O'_j,k) \cdot P_{co}(O'_j,k)$.

The individual probabilities in the definition of $P_{su}(M_j)$ are defined as follows. $P_{om}(O_j) = P_o^{|O_j|}(1-P_o)^{|O'_j|}$since it directly depends on P_o. For computing $P_{ti}(O'_j,k)$, we use $P_t(n,t)$ and assume that if x transmission faults occurs in the interval when messages in O'_j are transmitted, all messages are delayed by x retransmissions in the worst case. Due to space constraints, we omit a detailed timing analysis for this step, but note that the high-level approach is inspired by Broster et al.'s method [2]. For $P_{co}(O'_j,k)$, recall the aggregation protocol at B: if a majority of the k messages transmitted on or before $d_{a,j}$ are correct, then B infers the correct value as input. Thus, using P_o and assuming that l denotes the number of messages with corruptions: $P_{co}(O'_j,k) = \sum_{l=0}^{\lceil k/2\rceil-1} \binom{k}{l} \cdot P_c^l \cdot (1-P_c)^{k-l}$. Although the preceding definition pessimistically assumes that $P_{co}(O'_j,k) = 0$ if k is even and $l = k/2$, a more fine-grained analysis of such ambiguous cases is planned. Next, we conclude with a brief discussion of initial experiments and future work.

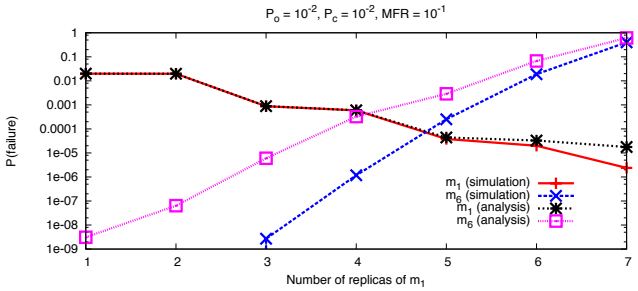


Fig. 2. Analysis and simulation of a set of six implicit-deadline messages $\{m_1,\ldots,m_6\}$, with periods $T_1 = 5ms$ and $T_2 \ldots T_6 = 10ms$. We assume that m_1 can be affected by an omission fault, a commission fault, or a transmission fault, whereas $m_2,\ldots,m_6$ are affected by transmission faults. Thus, only m_1 is replicated. Priorities are assigned using Davis and Burns's algorithm [3], which assigns the highest priorities to m_1 and its replicas. The Poisson distribution for transmission faults assumes a mean fault rate (MFR) of 0.1 faults per *ms*. Results for m_2–m_5 have been omitted to avoid clutter. The CAN bus schedule was simulated for 100,000 seconds and each message instance, *i.e.*, each $\{m_{i,k}\}_{i,k}$, was counted as a separate event. The entire simulation procedure was repeated 640 times.

IV. Example and Future Work

We simulated the transmission of a synthetic message set and compared the observed results with the analytical predictions (Fig. 2). For message m_1, which was replicated, the probability of failure reduced upon increasing its replication factor; whereas for message m_6, which was not replicated, a reverse trend was observed. The results corroborate the expected fault-tolerance versus timeliness tradeoff, and show that the analysis is accurate enough to predict similar trends. Another observation is that if the replicated task has a higher priority, the robust priority assignment algorithm [3] assigns high priorities to its replicas as well, compromising on the timeliness of the lower priority tasks.

In the future, we plan to develop a replication-aware priority assignment scheme for CAN bus messages. In addition, using the proposed analysis, we aim to answer questions such as: what is the optimal replication factor so that the probability of failure for each message is below a given threshold, how does a system's resiliency change when taken from lab conditions to harsher environments, *etc.* Currently, we are working on deriving aggregation procedures and analyses for hosts with asynchronous clocks and for sporadic messages.

References

[1] S. Borkar, "Designing reliable systems from unreliable components: the challenges of transistor variability and degradation," *Micro*, 2005.
[2] I. Broster, A. Burns, and G. Rodríguez-Navas, "Timing analysis of real-time communication under electromagnetic interference," *Real-Time Systems*, 2005.
[3] R. I. Davis and A. Burns, "Robust priority assignment for messages on controller area network (CAN)," *Real-Time Systems*, 2009.
[4] I. Noble, "EMC and the automotive industry," *Electronics & communication engineering journal*, 1992.
[5] K. Tindell, A. Burns, and A. J. Wellings, "Calculating controller area network (CAN) message response times," *Control Engineering Practice*, 1995.

Fault Tolerance in Real-Time Resource Partitions

Rachel Victoria Madrigal, Albert Mo Kim Cheng
Department of Computer Science
University of Houston
Houston, Texas, USA

***Abstract*—Fault tolerance is vital to real-time systems to ensure their reliability. This research is on the fault tolerance of hierarchical real-time systems. Then, we propose the placement of uniformly distributed checkpoints on Regularity-based Resource Partition Models.** [1]

I. Introduction

This paper describes our preliminary work on fault tolerance in real-time resource partitions. Reliability is crucial to a real-time system. In the event of a fault, it is necessary, not only to resume operation and deliver correct results, but also ensure that these are done within specific deadlines. Much research is done on fault tolerance in real time systems to meet such requirements.

One specific area of research within real-time systems is hierarchical scheduling. This occurs in open systems, wherein physical resources time-share different virtual partitions [1]. With this approach, a resource-level scheduler assigns partitions, while a task-level scheduler works only with the tasks themselves. Our research focuses on the schedulability of partitions and making these systems fault-tolerant.

Fault tolerance can be done through on-line fault detection, checkpointing, and rollback recovery. Checkpoints save the current state of the process. They are distributed throughout the program, either periodically or explicitly by the programmer. If a fault occurs, the system can then rollback to these checkpoints and reexecute. In real-time systems, the biggest concern is its impact on execution time.

The rest of this paper is organized as follows. Section II will discuss concepts needed to understand our approach. Section III will present our fault tolerance approach. Section IV will explain the further research that still needs to be done.

II. Established Concepts

A. Regularity-based Resource Partition Model

The Regularity-based Resource Partition Model [1] clearly follows scheduling separation. In this work, a real-time virtual resource operates at a fraction of the shared physical resources with a varying but bounded rate. A resource partition is a description of how to assign time intervals periodically on physical resources to a real-time virtual resource [2].

[1]Supported in part by the National Science Foundation under Awards No. 0720856 and No. 1219082.

Mok's algorithms were further improved with Li and Cheng's AAF-Multi [2] and Magic7-Pfair-Mixed Algorithm [3]. Before incorporating fault tolerance into this model, it is first necessary to review the concepts discussed in these papers.

Definition II.1. *A resource partition $\prod$ is a tuple (S, p), where $S = \{s_1, s_2, \ldots, s_n : 0 \leq s_1 < s_2 < \ldots < s_n < p\}$ is the time-slice sequence, and p is the partition period.*

Definition II.2. *$|S|$ represents the number of items in the sequence S.*

Definition II.3. *The Availability Factor of a resource partition $\prod = (S, p)$ is $\alpha(\prod) = \frac{|S|}{p} \epsilon (0, 1]$.*

The availability factor represents the fraction of time slices assigned to a partition.

Definition II.4. *The Supply Function $S(t)$ of a partition $\prod$ is the total number of time-slices that is available in $\prod$ from time 0 to t.*

Definition II.5. *The Instant Regularity $Ir(t)$ at time t on partition $\prod$ is given by $S(t) - t \cdot \alpha(\prod)$.*

Definition II.6. *Letting a, b, k be non-negative integers, the supply regularity $R_s(\prod)$ of partition $\prod$ is equal to the minimum value of k such that $\forall a, b, |Ir(b) - Ir(a)| < k$.*

Definition II.7. *Given a partition $\prod$ with availability factor α and supply regularity R_s, the Adjusted Availability Factor $AAF(\alpha, R_s)$ is the total of the availability factors of partitions that are used to compose $\prod$.*

Li and Cheng[3] provide the following simple recursive algorithm to compute for AAF.

$$AAF(\alpha, k) =$$

$$\begin{cases} 0 & if\ \alpha = 0; \\ \frac{1}{2^n}, where\, n = \left\lfloor log_{\frac{1}{2}} \alpha \right\rfloor & if\ \alpha > 0 \quad and\, k = 1; \\ AAF(\alpha - \mathcal{L}(\alpha), k-1) + \mathcal{L}(\alpha) & otherwise, \end{cases}$$

$$where\, \mathcal{L}(\alpha) = \frac{1}{2^n}, where\, n = \left\lceil log_{\frac{1}{2}} \alpha \right\rceil .$$

Theorem II.1. *Given a sequence* $\{(\alpha_i, k_i) : i = 1, 2, \ldots, n\}$ *where* α_i *is the availability factor and* k_i *is the supply regularity of a partition* $\prod_i$*, these partitions are schedulable on a single resource if* $\sum_{i=1}^{n} AAF(\alpha_i, k_i) \leq 1$.

B. Fault and Recovery Model

This research focuses on transient faults, which are temporal and subside quickly. They occur more frequently than permanent faults [4]. It is also assumed that a fault can be detected before the next checkpoint is created. Thus, if a fault occurs, rolling back to the previous checkpoint would be sufficient. The checkpoints may be stored in main memory rather than secondary storage, reducing overhead [5].

Based on Feng's explanations [6], tasks are only scheduled after the partition scheduling is done. Therefore, if checkpoints are scheduled to be performed before the tasks themselves start running, then deadlines should not be of concern. This becomes the responsibility of the task scheduler.

Since we are dealing with multiprocessors, it is also important to decide between coordinated and uncoordinated checkpointing. Coordinated checkpointing creates a global checkpoint with which the entire system may roll back to in case of a fault. On the other hand, uncoordinated checkpointing maintains different checkpoints for each processor (CPU), requiring tracking the interactions between the processors. The former consumes more energy, but the latter may lead to a domino effect. For this research, coordinated checkpointing is simpler and more feasible.

III. Research Work

A. Uniformly Distributed Checkpoints

One option in checkpoint placement is to distribute them uniformly. In order to schedule these checkpoints with the resource partitions, we may think of the checkpoints as partitions themselves.

Definition III.1. *Let* $p_{min} = \{min(p_i) : i = 1, 2, \cdots, n\}$ *where* p_i *is the period of partition* $\prod_i$*, and* n *is the number of partitions*

Given the periods of each resource partition, we know how often sets of deadlines will occur. We choose the minimum partition period to minimize the number of deadlines missed in case of a fault.

Definition III.2. *Let* c *be the worst-case cost of checkpointing and self-test of partition* $\prod_i$ *in terms of time-slices.*

Definition III.3. *A checkpoint resource partition is denoted by* ζ*, where* ζ *is a tuple* (S, p)*, where* $S = \{p_{min+1}, p_{min+2} \cdots, p_{min+c}\}$ *and* $p = p_{min} + c$.

Following the definitions in the previous section, it can be easily be stated that in a checkpoint resource partition, $|S| = c$ and $\alpha(\zeta) = \frac{c}{p}$. All time-slices for this partition will only be from p_{min+1} onwards since the checkpoints will be placed at the end of partitions. Thus, the slope of $S(t)$ may be separated into two values. First, if $\forall t \leq p_{min}$, then the slope of $S(t)$ is 0, and if $\forall t > p_{min}$, then the slope of $S(t)$ is 1. Simple algebra proves that from the definition of Supply Regularity (II.6) $k = \frac{|S|}{p}$. Therefore, $R_s = 1$.

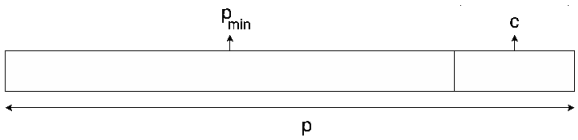


Figure III.1. Checkpoint resource partition

These computations allow us to modify Theorem II.1 [1] to incorporate checkpoints. Figure 3.1 provides a visual representation of a checkpoint resource partition.

Theorem III.1. *Given a sequence* $\{(\alpha_i, k_i) : i = 1, 2, \ldots, n\}$ *where* α_i *is the availability factor,* k_i *is the supply regularity of a partition* $\prod_i$*, and* n *is the number of partitions, these partitions are schedulable on a single resource with checkpoints if* $\sum_{i=1}^{n} AAF(\alpha_i, k_i) + AAF(\frac{c}{p}, 1) \leq 1$.

IV. Future Work

In this work, we have presented ideas on uniformly distributed checkpoints on a single resource. This is research that may still be extended. More work may be done on scheduling the checkpoints and testing to find the checkpoint execution time and rollback recovery time.

References

[1] A. K. Mok, X. Feng, and D. Chen, "Resource partition for real-time systems," in *Real-Time Technology and Applications Symposium, 2001. Proceedings. Seventh IEEE*. IEEE, 2001, pp. 75–84.

[2] Y. Li, A. M. Cheng, and A. K. Mok, "Regularity-based partitioning of uniform resources in real-time systems," in *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2012 IEEE 18th International Conference on*. IEEE, 2012, pp. 368–377.

[3] 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.

[4] R. Melhem, D. Mossé, and E. Elnozahy, "The interplay of power management and fault recovery in real-time systems," *Computers, IEEE Transactions on*, vol. 53, no. 2, pp. 217–231, 2004.

[5] S. M. Srinivasan, S. Kandula, C. R. Andrews, and Y. Zhou, "Flashback: A lightweight extension for rollback and deterministic replay for software debugging," in *USENIX Annual Technical Conference, General Track*. Boston, MA, USA, 2004, pp. 29–44.

[6] X. Feng, "Design of real-time virtual resource architecture for large-scale embedded systems," 2004.

A Scratchpad Memory-Based Execution Platform for Functional Reactive Systems and its Static Timing Analysis

Zeinab Kazemi, Albert M. K. Cheng
Department of Computer Science
University of Houston
Houston, Texas, USA
Email: zkazemialamouti@uh.edu, cheng@cs.uh.edu

***Abstract*—Priority-based Functional Reactive Programming (P-FRP) is a new variant of FRP to model reactive applications in real-time systems. In a P-FRP system, similar to a preemptive system, a higher-priority task can preempt a lower-priority one and make the latter abort. The lower-priority task will restart after the higher-priority tasks complete their execution. However, unlike the preemptive model, when a task aborts, all the changes made by the task are discarded. In previous studies, the value of Worst Case Execution Time (WCET) of each task is considered a priori. Besides, accurately determining the upper bound on the WCET requires considering memory latency, but this has been ignored in the previous work. In this work, we introduce a scratchpad memory-based platform to determine the WCET of tasks considering the memory cost of aborted tasks. We first compute WCET of a task in a P-FRP system, and then calculate the memory penalty caused by preemption. Preliminary experimental results using real-time tasks sets are also presented.**

I. Introduction

Functional Reactive Programming (FRP) is a framework for constructing reactive applications. FRP was originally developed based on Haskell language [3]. FRP exploits many of the characteristics of high-level languages such as abstraction, maintainability, and comprehensibility. These characteristics have encouraged the academics to take advantage of FRP in real-time systems. However, Haskell does not provide real-time guarantees and so the same applies to FRP. [4] introduces a new variation of FRP called Priority-Based FRP (P-FRP). In this model, while keeping the semantics of the language, a priority is assigned to each event. P-FRP is quite different from preemptive and non-preemptive systems. When a higher-priority event preempts the lower-priority one, the latter will be aborted and all of its changes will be discarded (*abort and restart*). The aborted event requires to start the execution from the beginning and if uninterrupted all the changes should be atomically committed. By using the P-FRP model, analysis of the responsiveness of tasks executing on such systems, will be more comprehensible. A polynomial algorithm to calculate the upper bound on response time in P-FRP has been presented by Belwal et al [2]. Although response time evaluation is critical for real-time systems, the results will not be reliable unless the WCET of each task and their memory access latency are computed. Thus, for computing the response time, the challenge is to compute the upper bound of total computation time of each task and the latency of its memory operations.

* Supported in part by the National Science Foundation under Awards No. 0720856 and No. 1219082.

II. WCET on Functional Reactive Systems

We defined the term functional reactive systems as systems based on P-FRP. The behavior of tasks in such systems is quite similar to memory transactions. A task is either committed or discarded. Therefore P-FRP and functional reactive systems benefit from lack of task side effects, checkpointing, rollback cost, and provide type safety. In order to estimate the WCET of a task, we need to have prior knowledge of micro-architectural specifications of the system, on which, task is executing. Since functional reactive systems have not been fully implemented or even described, one needs to specify a certain platform for the system. The challenge is to develop a model for these systems, using the available micro-architectures to fit the properties of P-FRP. In this on-going work, we design and describe a platform applicable for P-FRP characteristics.

One crucial matter to consider for abort-and-restart model is that, after preemption of each task, all the changes made by the preempted task should be discarded. For this end, a task should not commit anything to memory while executing, instead it should update the changes to a temporary location or volatile registers. Besides, all the memory write operations need to be at the end of each task. If otherwise, the task can be preempted after the write operation and this will violate the P-FRP requirements. To address this problem, we use on-chip Scratchpad Memory (SPM), to store the in-progress work, and then commit all the changes made by a task to off-chip memory. SPM benefits from predictable timing behavior and is a suitable fast on-chip memory for calculating the WCET. Despite Caches that are controlled by hardware, SPM can be managed in software level. SPM lets programs request for space allocation, loads their data from main memory, and commits the changes back to the memory. Although hardware management of cache makes it a desirable choice for general systems, it leads to unpredictable timing behavior. While using software manageable SPM provides predictable access memory latency which makes it a better alternative for real-time systems [8].

III. Scratchpad memory in P-FRP systems

Since in our study, P-FRP systems are generally embedded real-time systems, and in such systems, lower-priority tasks may frequently get preempted by higher-priority ones, these tasks should be as small as possible for the system to have a minimal overhead. For this reason, we presumed that each task is small enough to fit into SPM. In SPM, read and write from/to main memory are handled by Direct Memory Access (DMA) to minimize the latency. If the task is already in SPM, then there is no need to read the code again. However, it is required to load the input data from SPM because the input data may have been changed by higher-priority tasks. However, if it is not located in SPM, using DMA, the task and its data should be read from memory. In addition, committing data to memory should be atomic to avoid side effects. Thus, DMA write operation is required to have the highest priority so it can commit to memory without any preemptions.

IV. Safe upper-bound on WCET

In order to calculate the upper-bound, we changed the current WCET solutions to adapt it to a P-FRP system. For implementing this system, we will take a task and hardware parameters as inputs and return the WCET as the final output. For this end, the upper bound on execution of each basic block is calculated. In the next stage by using Integer linear Programming (ILP), we find the WCET estimate of the program. We calculate the WCET of our P-FRP system based on the extension of ILP objective function introduced in Chronos, an open source static timing analyzer [6]. For this purpose, we have to propose some changes: (1) in our FRP system instead of cache we use SPM for the reasons discussed before. (2) each task is small enough to fit into the SPM when it starts its execution. The modifying ILP objective function will contain DMA's read latency which depends on the availability of the task in SPM, DMA's write latency, and sum of the execution times of all basic blocks in the task's code which depends on the results of branch predictions of previous basic blocks in the execution path.

V. Preliminary results

We evaluate our approach by comparing the actual execution time of each task with our estimated WCET using the modified ILP objective function. In order to compute the actual WCET of each task, we use the SimpleScalar/PISA which is a computer architecture simulator developed by Todd M. Austin [1]. We use simplesim3.0e version using PISA instruction set. The memory latency and the cost of DMA operations caused by preemption, are also added to the simlescalar source code to adapt it to the P-FPR. We have selected 8 benchmarks from the SNU real-time benchmark set [7] to compare the values of simulation time and the estimated WCET. Table I indicates the comparison of simulation and WCET estimation of selected benchmarks (tasks). The data read from memory to SPM, and data committed back to memory in Bytes can be observed in Table I. In this work, we assume that SPM size is limited to only one task. Every time task τ arrives, its data and code should be loaded to SPM. To evaluate the accuracy of the estimated WCET, we define the term *AccuracyRatio*. AccuracyRatio, for each task τ is calculated as follows:

$$AccuracyRatio = Estimation_\tau / Simulation_\tau$$

TABLE I. Simulation of actual execution time compared with WCET estimation of real-time SNU benchmarks in a SPM-based P-FRP system

Tasks	Simulation (cc)	Estimation (cc)	Read Data (B)	Committed Data (B)	Memory Latency (cc)	Ratio
fib	145	155	164	4	71	1.07
cnt	5362	6546	840	416	570	1.22
insertsort	1142	1518	496	44	245	1.33
crc	22213	23666	1108	4	504	1.07
matmul	3241	5152	752	144	406	1.59
qurt	1692	1780	1072	32	501	1.05
bs	366	553	380	4	173	1.51
jfdctint	4560	4580	2952	256	1457	1.00

As expected, the upper-bound on the WCET of each task should be greater than the actual execution time, resulting in an AccuracyRatio greater than 1. The average AccuracyRatio for the evaluated benchmarks is 1.23. The maximum AccuracyRatio is 1.59 and the minimum is almost 1. The reason for this close estimation is that loop bounds and user constraints used for the ILP objective function, are accurately selected. Although depending on the benchmark source code, determining accurate constraints could be time consuming.

VI. Ongoing work

In this work, we have assumed that SPM can only contain one task at a time; this leads to a pessimistic estimation of the WCET since if multiple tasks could fit into the SPM, there is no need to load the codes of tasks already available in SPM. We are currently working on a system with a larger SPM with the space capacity for multiple tasks. Availability of tasks in SPM can result in shorter memory read latency, and less pessimism, thus, a tight WCET upper bound. Furthermore, in the described model, it is assumed that a task is small enough to fit into SPM. Currently, we are scaling our implementation to support larger tasks. To address this issue, we are working on a dynamic code management on SPM to dynamically load part of task's code and data to memory [5]. We are also modifying the upper bound on response time algorithm presented by Belwal et al to adapt it for a variable task computation time and therefore to evaluate it considering memory penalty calculated in previous sections.

References

[1] Todd Austin, Eric Larson, and Dan Ernst. Simplescalar: An infrastructure for computer system modeling. *Computer*, 35(2):59–67, 2002.

[2] Chaitanya Belwal, Albert MK Cheng, and Yuanfeng Wen. Response time bounds for event handlers in the priority based functional reactive programming (p-frp) paradigm. In *RACS*. ACM, 2012.

[3] P. Hudak and J. Fasel. A gentle introduction to Haskell. *ACM SIGPLAN Notices*, 1992.

[4] Roumen Kaiabachev, Walid Taha, and Angela Zhu. E-frp with priorities. In *Proceedings of the 7th ACM & IEEE international conference on Embedded software*. ACM, 2007.

[5] Yooseong Kim, David Broman, Jian Cai, and Aviral Shrivastaval. Wcet-aware dynamic code management on scratchpads for software-managed multicores. In *RTAS*, 2014.

[6] Xianfeng Li, Yun Liang, Tulika Mitra, and Abhik Roychoudhury. Chronos: A timing analyzer for embedded software. *Science of Computer Programming*, 2007.

[7] SNU. Homepage of snu real-time benchmark suite, 2014.

[8] Vivy Suhendra, Tulika Mitra, Abhik Roychoudhury, and Ting Chen. Wcet centric data allocation to scratchpad memory. In *RTSS*. IEEE, 2005.

Towards an Interpretation of Mixed Criticality for Optimistic Scheduling

Marcus Völp, Michael Roitzsch, Hermann Härtig

Technische Universität Dresden
Dresden, Germany 01062
Email: ⟨name.surname⟩@tu-dresden.de

***Abstract*—Mixed criticality systems promise significant reductions in weight, energy, and costs by consolidating safety-critical tasks with different certification requirements into one or a few systems. The ultimate goal is to preserve independently established guarantees at the individual certification levels while combining tasks and allowing them to share resources—most notably the CPU—within and across criticality levels. In this regard, mixed criticality scheduling can be seen as a mechanism for orchestrating the pessimism of assurance requirements and the resource overapproximation it entails. However, in our opinion, the scope of mixed criticality scheduling is much wider. We see opportunities to apply and reuse the frameworks and theory of mixed criticality in a much wider setting and for a variety of different application scenarios. In this work-in-progress report, we would like to share our vision of applying mixed criticality to orchestrate optimism and to control degradation in situations where developers had been too optimistic.**

I. Introduction

In 2007, Vestal [1] introduced the notion of mixed or multi-criticality to characterize systems, which consolidate tasks with varying degrees of execution time assurance. His groundbreaking insight was to arrange schedulability conditions such that if a more critical task fails to meet the requirements imposed on less critical tasks, it can still meet the requirements at higher levels, including its own, though possibly at the cost of violating the guarantees given to tasks that are less critical for the proper operation of the system. First, Vestal expressed these requirements only for worst-case execution times by requiring $C_i(l) \leq C_i(h)$ for any two criticality levels $l < h$ where $l, h \in L$. Subsequent works [2]–[4] extended his approach also to minimal inter-release times $T_i(l) \geq T_i(h)$ and in part also to deadlines $D_i(h) \leq D_i(l)$ [3]. The assumptions are that the set of criticality levels L is small and that schedulability conveys hard real-time guarantees provided the rely condition at the respective criticality level is met. More precisely:

Definition 1 (Schedulability)**.** *A task τ_i is* schedulable *in the sense that all jobs $\tau_{i,j}$ receive $C_i(l_i)$ time in between the release $r_{i,j}$ of this job and its deadline $r_{i,j} + D_i(l_i)$, provided the rely condition holds that higher criticality jobs $\tau_{h,k}$ (with $l_h > l_i$) complete within $C_h(l_i)$ time and that they do not arrive more frequent than $T_h(l_i)$.*

In this paper, we take the view that mixed-criticality is not only beneficial when orchestrating pessimism but also to express which threads to degrade in performance if optimistic assumptions about the system behavior cease to hold. We introduce our optimistic interpretation of mixed criticality (Sec. II), sketch early ideas how to parameterize optimism (Sec. III), and review application areas beyond the realm of safety-critical settings (Sec. IV).

II. Towards an Optimistic Interpretation of Mixed Criticality

In the early days, priority expressed importance and a low prioritized thread was only considered for running after all higher prioritized threads completed or if they were blocked waiting for resources or other events. However, since the seminal work of Liu and Layland [5], priorities stopped to assume this role and degraded to parameters, which originate from an admission test. The least important thread is not necessarily the one with a deadline in the most distant future or with the largest period.

As first realized by Graydon and Bate [6], criticality may assume this role. When switching from a low-criticality mode to a higher-criticality mode, excess budgets are granted, which may violate the guarantees of low-criticality threads, dropping them to a lower priority or reducing their budget to the slack that remains in this higher-criticality schedule. We write $\tau_i \prec^h \tau_j$ to denote that task τ_i should be sacrificed before τ_j if τ_j requires the additional time provided by $C_j(h)$ and refine this relation to jobs $\tau_{i,m}$ if more fine-grained degradation control is required[1]. Like with criticality levels, we drop all threads τ_k with $\tau_k \prec^h \tau_j$ in case τ_j needs to tap into $C_j(h)$. There are a few immediate consequences: (i) the partial ordering of $\prec^h$ makes degradation is more selective (a low-criticality job τ_l is not affected from a budget overrun of a higher criticality thread τ_h unless $\tau_l \prec^h \tau_h$); (ii) criticality levels may no longer be static but may follow some state or job dependent pattern; (iii) topological sort by successive removal of not dominated tasks re-establishes the classical criticality levels, which are a pessimistic bound of $\prec$; and, most notably, (iv) the vector of execution time estimates C_i must no longer consist of estimates of the worst-case behavior but may include monotonically increasing sequences of estimates, optimistic and pessimistic ones.

Low criticality may represent best effort or soft real-time modes. Unlike classical approaches, where these tasks are often scheduled in the background or at low priority levels, mixed criticality allows integrating these tasks at the priority where their completion can best be guaranteed while still

[1] Thanks to reviewer 2 for pointing out this possibility.

retaining the option of dropping them in case their resources are needed for more important jobs. Of course this assumes that the scheduling algorithm is capable of tolerating criticality inversion. High criticality on the other hand may remain for hard real-time, safety critical threads.

III. Parameterizing Optimism

Reusing the large body of mixed criticality reasoning and scheduling frameworks prevents us from deviating significantly from the standard taskset formalization. Hence, we assume as usual that tasks τ_i ($i \in [0, \dots n]$) are represented as $\tau_i = (l_i, C_i, T_i, D_i)$ where C_i is a vector of criticality dependent execution time estimations and T_i and D_i are similarly dependent periods and deadlines. However, unlike Vestal's model, we assume C_i to be merely the budgets with no required connection to worst-case execution times, etc.

Decoupling budgets from WCET estimates allows us to consider quality and confidence and derive budgets from the execution time distribution in a more optimistic way. For example, given a quality $\mathcal{Q}(C_i)$ and a priority ordering, QAS [7], [8] derives a budget to guarantee that the likelihood of completing τ_i's jobs before their deadlines is $\mathcal{Q}(C_i)$. Except for trivial cases, these budgets are well below the $\mathcal{Q}(C_i)$-quantiles of the execution time distribution. In the future, we plan to include confidence $\mathcal{C}(C_i)$ in the estimates in a similar fashion.

IV. Application Areas

Opening up mixed-criticality scheduling for optimistic assumptions about resource usage widens the application areas of existing frameworks. Instead of relying on upper bounds of expected execution times, resource estimation and prediction can be employed. Such architectures enable optimistic scheduling close to the actual resource usage [9], but due to the possibility of underestimations are currently unsuitable for hard real-time work. Combined with a mixed-criticality policing mechanism, such optimistic scheduling regimes can ensure that understimations never harm highly critical work.

Other use cases we envision, where mixed-criticality enables more optimistic resource allocation include fault-tolerant systems. There, the optimistic assumption is a fault-free run, but when a malfunction is detected, highly critical work is recovered preferentially. Hardware features like transactional memory, which improve average case, but not worst-case performance can also be utilized more effectively. Optimistically assuming transactions to complete allows us to reap their benefits without sacrificing the safety of critical work under the pessimistic assumptions that resources are contended.

V. Related Work

Although, to the best knowledge of the authors, this is the first work to apply mixed-criticality scheduling to non-safety critical setups, others have questioned its limited scope before. For example, Graydon and Bate [6] were first to realize that even in the context of safety-critical systems, not all notions of criticality meant the same thing. They propose to distinguish confidence, importance (i.e., the consequence of overrunning deadlines), service mode and mode of operation. Our interpretation exploits criticality levels to express importance. However, in the notion which task to degrade rather than just covering the consequence of deadline misses. Ekberg et al. [10], [11] extends mixed-criticality scheduling with a limited number of criticality levels to arbitrary numbers of modes and organize switching between them with directed acyclic graphs. Burns [12] related mixed-criticality mode switching to more general modes.

VI. Conclusions

In this work, we have raised the question whether and how mixed-criticality systems can be transformed from orchestrating the pessimisim of differing assurance requirements in safety-critical systems to more optimistic approaches where C_i is no longer an estimate of the worst case behavior. Like Graydon and Bate before, we have seen that separating confidence in an estimation from the effect of deadlines overruns allows us more fine-grained control when degrading tasks at low criticality levels. And although further research is required to pursue this direction, mixed criticality systems benefit from optimism.

Acknowledgment

The work is in part funded by the German Research Council through the cluster of excellence Center for Advancing Electronics Dresden.

References

[1] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in *Real-Time Systems Symposium*. Tucson, AZ, USA: IEEE, December 2007, pp. 239–243.

[2] S. Baruah, "Certification-cognizant scheduling of tasks with pessimistic frequency specification." in *7th IEEE International Symposium on Industrial Embedded Systems (SIES12)*, 2012, pp. 31–38.

[3] S. Baruah and A. Burns, "Implementing mixed criticality systems in ada." in *Reliable Software Technologies - Ada-Europe*, A. Romanovsky, Ed. Springer, 2011, pp. 174–188.

[4] A. Burns and S. Baruah, "Timing faults and mixed criticality systems." *Dependable and Historic Computing (LNCS 6875)*, pp. 147–166, 2011.

[5] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, January 1973.

[6] P. Graydon and I. Bate, "Safety assurance driven problem formulation for mixed-criticality scheduling," in *1st Workshop on Mixed Criticality Systems*, 2013, pp. 19–24.

[7] C.-J. Hamann, J. Löser, L. Reuther, S. Schönberg, J. Wolter, and H. Härtig, "Quality Assuring Scheduling - Deploying Stochastic Behavior to Improve Resource Utilization," in *22nd IEEE Real-Time Systems Symposium (RTSS)*, London, UK, December 2001.

[8] M. Völp, "What if we would degrade lo tasks in mixed-criticality systems?" in *20th IEEE Real-Time and Embedded Technology and Applications Symposium - Work in Progress Session (RTAS-WIP 2014)*, Berlin, Germany, April 2014.

[9] M. Roitzsch, S. Wächtler, and H. Härtig, "Atlas: Look-ahead scheduling using workload metrics," in *19th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2013)*, April 2013.

[10] P. Ekberg and W. Yi, "Bounding and shaping the demand of generalized mixed-criticality sporadic task systems," *Journal of Real-Time Systems*, vol. 50, pp. 48–86, 2014.

[11] P. Ekberg, M. Stigge, N. Guan, and W. Yi, "State-based mode switching with applications to mixed criticality systems," in *1st Workshop on Mixed Criticality Systems*, 2013, pp. 61–66.

[12] A. Burns, "System mode changes - general and criticality-based," in *2nd Workshop on Mixed Criticality Systems*, 2014, pp. 3–8.

From TIMES to TIMES^{++}

S.M. Jakaria Abdullah*, Pontus Ekberg*, Nan Guan†, Martin Stigge* and Wang Yi*
*Uppsala University, Sweden
†Northeastern University, China

***Abstract*—TIMES [1] is a tool suit for scheduling analysis of complex real-time systems with powerful features for modeling, but it suffers from state-space explosion problem inherent to automata-based models. The Digraph Real-Time (DRT) task model [2] is a strict generalization of earlier proposed graph-based real-time task models and is also one of the most expressive task models with tractable feasibility analysis. We are developing a new version of TIMES with the core input modeling language being replaced with DRT, the new analysis engine based on efficient algorithms for DRT models developed in a series of recent works [3][4][5] and additional features for system-level and component-level analysis.**

I. Background

Modern real-time systems are composed of non-trivial functionalities represented by complex tasks distributed across different components. Choosing appropriate abstractions for task functionality in analysis is a fundamental problem for real-time system designers. Using a simplified task model in component-level analysis forces system designer to abstract away important software details, like a local loop inside a task. On the contrary, use of sophisticated automata or graph-based task models can make both component and system-level analysis intractable due to the increased complexity of the representation. As a result, existing system analysis tools like MAST [6] and SymaTA/S [7] use less expressive task models in component-level analysis. This means precision of system-level analysis by these tools can be significantly reduced by less expressive component-level task abstractions.

TIMES [1] is a tool suit for schedulability analysis of complex real-time systems, in which task systems are modeled with task automata [8] and the analysis problems are solved using the UPPAAL model checker [9]. The TIMES modeling language based on task automata provides powerful expressiveness, but also limits the scalability of the analysis due to the state-space explosion problems. Recently Stigge et al. [2] showed that a graph-based task model called the *Digraph Real-Time (DRT)* task model is among the most expressive models with an exact pseudo-polynomial time feasibility test on preemptive uniprocessors. While DRT seems like an ideal replacement for task automata to make TIMES more scalable in component-level analysis, we want to extend TIMES for system-level analysis. *Finitary Real-Time Calculus (FRTC)* [10] is a system-level analysis technique with pseudo-polynomial time complexity which together with DRT analysis on the component-level can make TIMES analysis both scalable and tighter. The goal of this paper is to present our ongoing work on development of a new version of TIMES tool, called TIMES^{++}, utilizing DRT and FRTC techniques. First we present the modeling language of our tool and then we outline the features and architecture of the proposed software.

II. TIMES^{++}

A. Core Modeling Language

We use DRT tasks [2] as the core input language to describe the workload of a system. A DRT task set $\tau = \{T_1, \ldots, T_N\}$ consists of N independent tasks. A task T is represented by a *directed graph* $G(T)$ with both vertex and edge labels. The vertices $\{v_1, \ldots, v_n\}$ of $G(T)$ represent the types of all the jobs that T can release and are labeled with ordered pairs $\langle e(v_i), d(v_i)\rangle$ of non-negative integers, denoting worst-case execution-time demand $e(v_i)$ and relative deadline $d(v_i)$ of the corresponding job. The edges of $G(T)$ represent the order in which jobs generated by T are released. Each edge (u, v) is labeled with a positive integer $p(u, v)$ denoting the minimum job inter-release separation time. A job sequence $\rho = (J_1, J_2, \ldots)$ is generated by T if there is a path $\pi = (v_1, v_2, \ldots)$ through $G(T)$ such that for each job $J_i = (r_i, e_i, d_i)$ it holds that $r_{i+1} \geqslant r_i + p(v_i, v_{i+1})$, $e_i \leqslant e(v_i)$, and $d_i = r_i + d(v_i)$ where r_i, e_i and d_i denote release time, worst-case execution time and absolute deadline of job J_i respectively.

B. Component-Level Analysis Features

DRT as a modeling language offers efficient algorithms for tractable analysis problems and also algorithms which can solve typical instances of intractable problems. For example, Feasibility analysis for DRT task systems on uniprocessors is of pseudo-polynomial run-time complexity [2]. It is based on an iterative graph exploration procedure that uses a novel path abstraction technique. This complexity result is also valid on schedulability analysis using EDF. Static priority (SP) schedulability of DRT tasks is strongly *coNP*-hard [3]. However, an iterative refinement technique called *Combinatorial Abstraction Refinement* is shown to efficiently solve typical instances of the SP schedulability problem [4]. Exact response-time analysis of DRT tasks for SP and EDF schedulers is presented in [5]. It is also based on the combinatorial abstraction refinement framework to achieve exact results from initial over-approximations.

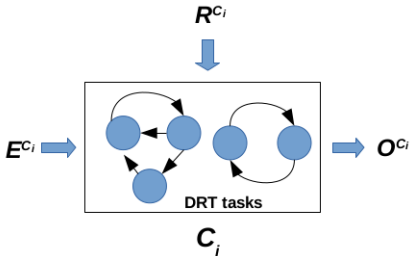


Fig. 1. A basic component dealt with TIMES^{++}

C. Extension to System-Level Analysis

A *system* S consists of a finite number of components, $S = \langle C_1, \cdots, C_n\rangle$. Each component is defined by a set of DRT tasks (see Figure 1). A component C_i receives a set of input signals $E^{C_i} = \{e_1^{C_i}, \cdots, e_m^{C_i}\}$ which can be generated by either dataflow or control flow events. DRT jobs corresponding to the specific input events will process the workload of the component and generate corresponding output event (control or dataflow) signals $O^{C_i} = \{o_1^{C_i}, \cdots, o_r^{C_i}\}$ depending on the context. We assume C_i will receieve processing resource R^{C_i} from a global server P. Depending on the input signals,

different sequences of DRT jobs execute inside a component with different resource requirements. If P supplies resource in a fixed budget manner meeting the worst-case requirement of the component, then we need a mechanism to compute remaining unused resource out of the component (see Figure 2). Alternatively, P can monitor arrival of input signals to a component to determine the required supply dynamically. Another option is to define component-wise modes based on patterns of input signal arrivals and their corresponding DRT jobs, then P only needs to monitor the mode to determine the required resource. Concrete procedures for this system level resource distribution mechanisms by P is part of our ongoing work.

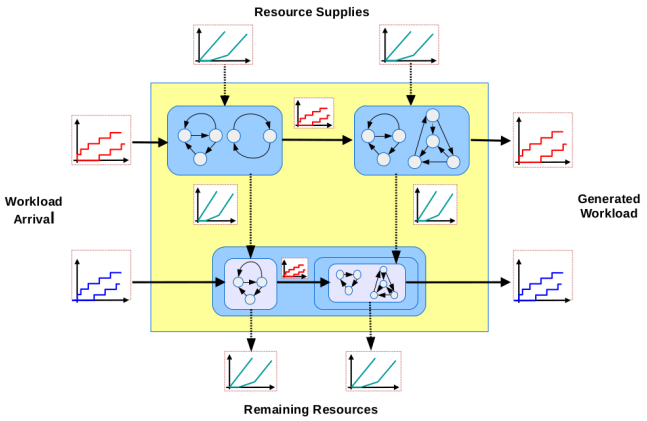


Fig. 2. Hierarchical structure of system-level models dealt with TIMES^{++}, where workload and resource supplies are represented by FRTC arrival curves and service curves respectively [10].

A few relevant outstanding research challenges include:

- How to transform dataflow applications into DRT tasks?
- How to schedule DRT components in a multiprocessor platform?
- How to synchronize DRT components?
- How to synthesize real-time code from DRT components?

III. Tool Design Overview

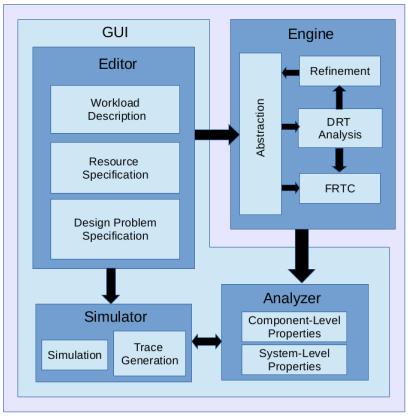


Fig. 3. TIMES^{++} Tool Architecture

Our tool consists of two main parts, a Graphical User Interface (GUI) and an analysis engine. The modular architecture of the tool, depicted in Figure 3, allows independent development of the components. Different tool components use XML to represent the system descriptions both internally and externally. The main components of this tool consist of:

Editor to graphically model a system and the abstract behaviour of its environment. Workload is modeled as set of DRT tasks, and different other real-time workload models have efficient transformation to a DRT task set. System resources are modeled with some abstraction of processing and communication capacities, such as FRTC service curves. The users can choose the setting of the design problem such as the scheduling policy.

Analysis Engine consists of four parts. The Abstraction module transforms the DRT workload models into abstract representations such as request bound functions (RBF) [4] which can be processed by the DRT analysis algorithms. The Analysis module is the core of the engine, which encapsualtes the DRT analysis algorithms for the specific design problem which are tractable. For intractable problems, the Refinement module uses the refinment framework [4] to iteratively obtain tighter and tighter analysis results from over-approximations. Finally, the FRTC module uses results from component-level analysis to determine system-level properties.

Simulator to visualize the system behavior as Gant charts and message sequence charts. The simulator can be used to randomly generate possible execution traces, or alternatively the user can control the execution by selecting the transitions to be taken. The simulator can also be used to visualize error traces produced in the analysis phase.

Analyzer to check various component-level and system-level properties of the system model including feasibility, scheduability, worst-case response time (WCRT), end-to-end delay and buffer requirement.

IV. Conclusions

Currently we are developing the GUI part of the tool and extending the existing analysis engine. Some interesting developments include DRT analysis with task synchronization and buffer capacity analysis for the analysis engine. We are also working on connecting DRT analysis with FRTC [11].

References

[1] T. Amnell, E. Fersman, L. Mokrushin, P. Pettersson, and W. Yi, "Times - a tool for modelling and implementation of embedded systems," in *Proc. of TACAS 2002*, pp. 460–464.

[2] M. Stigge, P. Ekberg, N. Guan, and W. Yi, "The Digraph Real-Time Task Model," in *Proc. of RTAS 2011*, pp. 71–80.

[3] M. Stigge and W. Yi, "Hardness Results for Static Priority Real-Time Scheduling," in *Proc. of ECRTS 2012*, pp. 189–198.

[4] ——, "Combinatorial Abstraction Refinement for Feasibility Analysis," in *Proc. of RTSS 2013*, pp. 340–349.

[5] M. Stigge, N. Guan, and W. Yi, "Refinement-Based Exact Response-Time Analysis," in *Proc. of ECRTS 2014*, pp. 143–152.

[6] M. Gonzalez Harbour, J. Gutierrez Garcia, J. Palencia Gutierrez, and J. Drake Moyano, "Mast: Modeling and analysis suite for real time applications," in *Proc. of ECRTS 2001*, pp. 125–134.

[7] A. Hamann, M. Jersak, K. Richter, and R. Ernst, "Design space exploration and system optimization with SymTA/S - Symbolic Timing Analysis for Systems," in *Proc. of RTSS 2004*, pp. 469–478.

[8] E. Fersman, P. Krcal, P. Pettersson, and W. Yi, "Task automata: Schedulability, decidability and undecidability," *Inf. Comput.*, vol. 205, no. 8, pp. 1149–1172, 2007.

[9] K. G. Larsen, P. Pettersson, and W. Yi, "Uppaal in a nutshell," *International Journal on Software Tools for Technology Transfer (STTT)*, 1997.

[10] N. Guan and W. Yi, "Finitary real-time calculus: Efficient performance analysis of distributed embedded systems," in *Proc. of RTSS 2013*, pp. 330–339.

[11] N. Guan, Y. Tang, Y. Wang, and W. Yi, "Delay Analysis of Structural Real-Time Workload," in *Proc. of DATE 2015*, to appear.

Seamless Requirements Flow Down from System to Software Components in Embedded Systems

Dr. Michael Winokur
Corporate Operations, IAI
POB 70100
Ben-Gurion Intl. Airport
+972-3-9353716
mwinokur@iai.co.il

Zvi Lando
Corporate Operations, IAI
POB 70100
Ben-Gurion Intl. Airport
+972-3-9355060
zlando@iai.co.il

Alon Modai
Corporate Operations, IAI
POB 70100
Ben-Gurion Intl. Airport
+972-3-9356369
amodai@iai.co.il

ABSTRACT

In this paper, we present WIP for a novel way to bridge the gap which hinders the seamless flow of requirements from system components to its constituent software components.

1. INTRODUCTION

There is a deep gap between systems engineers and software engineers in the way they analyze, understand, organize and allocate requirements. In developing a complex system, it often takes time for requirements to reach the various software components, leaving software engineers to base their work on a pile of guesswork disguised as "experience".

This paper describes a novel, systematic flow down process to bridge this gap, by seamlessly deriving software requirements from system requirements using a model based approach. For simplicity, the process will be illustrated using the Automatic Teller Machine (ATM) example used across the world for Embedded Systems courses [1].

2. REQUIREMENTS ENGINEERING PROCESS

The standard process views the systems as a hierarchical arrangement of nodes where each node represents:

- A set of stakeholder requirements for the node
- A conceptual model of the node
- A set of derived system requirements for the node
- Several sets of requirements, each allocated to a conceptual subsystem of the node (Output to the next level)

This method is particularly suitable for embedded Real Time systems where the requirements engineering process is repeated recursively at each node illustrated in Figure 1.

2.1 Deriving System Requirements

System Requirements express in solution-space terms what Stakeholder Requirements express in problem-space terms. Although in some cases derivation may be trivial, other cases require a precise engineering analysis in order to come up with a set equivalent to the original Stakeholder Requirement.

2.2 Kicking off Subsystems

Each subsystem resulting from system decomposition becomes the focus of working on a sub-node. The starting point for a sub-node consists of the following:

- The subsystem as a Black box, its input flows and output flows, and its environmental systems.
- The aggregate of all capabilities assigned to the subsystem by all dynamic processes it participates in.
- The set of requirements, both functional and non-functional like RT performance and time constraints, allocated to the subsystem and its functions.
- The set of events and conditions governing the invocation of the subsystem capabilities by all processes it participates in.

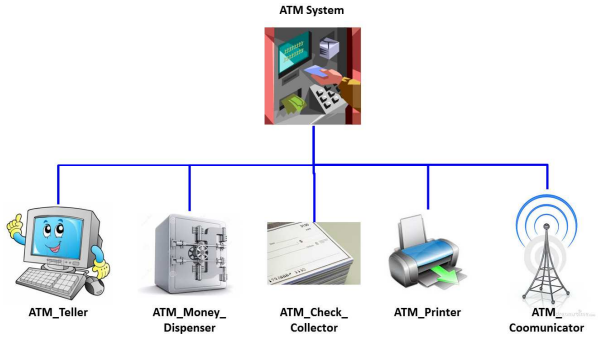


Figure 1: Automatic Teller Machine (ATM) example

2.3 The Software Requirements Model

The Software Requirements model is a set of information containing everything a software engineer needs to know in order to proceed to the Analysis and Design activities. The interesting point illustrated in the this section is that established model based software requirements methods reveals, at best, a gap of terminology and methodology with systems requirements modeling. In the worst case, software requirements models seem to start as an exploratory activity performed by the software engineers relying on written textual requirements or verbally communicated requirements from system engineers without relying on any developed system model [6].

The Software Requirements model consists of the following views:

- The Use-Case view
- The Interface view
- The States & Modes view
- Software Item Supplementary Requirements

3. FROM SYSTEM TO SOFTWARE REQUIREMENTS

This section presents the central subject of the paper: A systematic process for seamless flow between the system requirements model and its SW requirements model, as noted above. To achieve this, we propose the following steps:

3.1 Software Item Identification

Define as a software item only a (software) child node of some (system) parent node. For example, let us look at the Communicator Software Item of the ATM.

3.2 Use-Case Identification

Examine each Process of the ATM in which the software item Communicator participates. Example: The Withdrawing Cash process involving many HW and SW components shown in Fig. 1, alongside with the Communicator SW. Within this process:

1. Identify capabilities required of the software item
2. Model each required system capability satisfied by the Communicator as a Use-Case of this software item
3. Identify other SW and HW items of the parent node interacting with each Use Case of the Communicator
4. Model each such entity as an Actor
5. Model the interaction of the Actor with the Use-Case as either "Initiating" (arrow points to Use-Case) or "Participating" (Arrow points to Actor).
6. Allocate timing (RT) and other performance requirements to the Use-Case

An example of the resulting Use-Case diagram for the Communicator Software Item is shown in Figure 2. Note that the ATM Controller is the initiator of both Use-Cases.

3.3 Interface Identification

From the same ATM Process Withdraw Cash:

1. Identify flows into and from each capability
2. Add data items specified by each flow to the Interface Diagram View of the software item

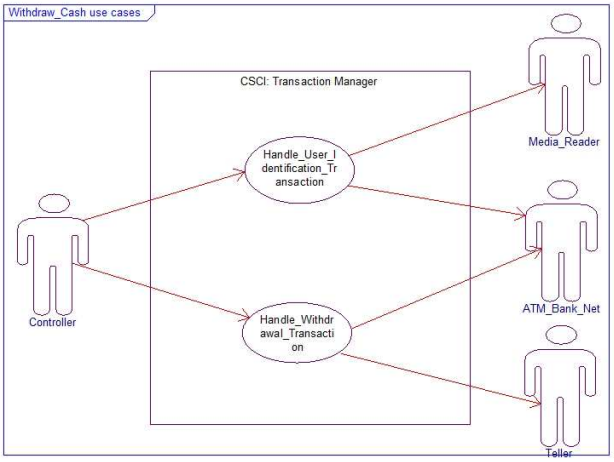


Figure 2: Use-Case Diagram for the Communicator

Example: A resulting sample Interface Diagram for a Communicator Use-Case (See Figure 3). Note that the full Communicator Interface View is comprised of several such Interface Diagrams.

3.4 Use-Case Investigation

For each identified Use-Case, investigate in collaboration with the system engineer and document the following in the Full Description of the Use-Case [3]

1. Any pre-conditions pertaining to the Use-Case
2. Any post-conditions required of the Use-Case
3. The nominal, exception and alternative interactions
4. Any additional requirements the Use-Case must satisfy

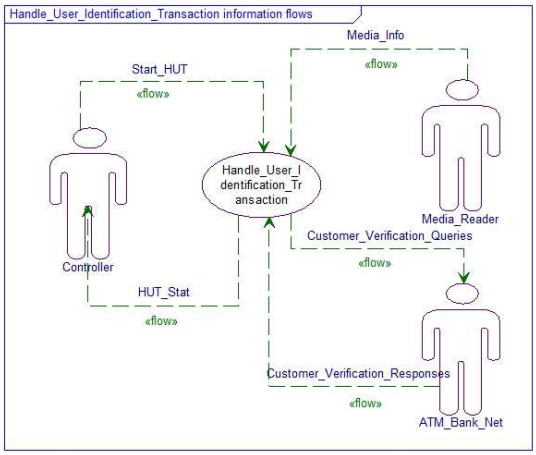


Figure 3: Interface Diagram for a Communicator Use-Case

3.5 Required States and Modes

If state & modes are required, model them [2].

3.6 Supplementary Requirements

Examine the set of requirements allocated to the software item.

1. Use any functional or performance requirement to refine a Full Use-Case Description.
2. Enhance the set of Supplementary Requirements with time reserves to satisfy the RT requirements for each Use-Case

3.7 Software Requirements Traceability

Treat each of the following Use-Case parts as a single software requirement and trace it to the requirements allocated to the SW item.

- The entire main success flow
- One exception flow
- One alternative flow
- The set of post-conditions

4. CONCLUSIONS

In this paper we presented a systematic process for seamless flow between the system requirements and the software requirements model of a CPS. We have shown how to bridge the seemingly incompatible functional system model and UML object oriented software model which are most commonly practiced, and appealing to each of the two communities. Moreover, the method also deals with the systematic flow down of no functional requirements, often disregarded by MBRE methods. Further research has been initiated on a quantitative assessment of the requirements engineering process presents as used in these real life projects including the number of stakeholders requirements (generally in the high hundreds to low thousands); model size, team sizes and development time and effort with comparative studies to previous processes. The results are expected to be submitted to the 2016 conference.

5. REFERENCES

[1] http://en.wikipedia.org/wiki/Automated_teller_machine

[2] David Harel and Michal Politi. *Modeling Reactive Systems with Statecharts*. McGraw-Hill (1998)

[3] Ivar Jacobson, Ian Spence, Kurt Bittner, Use-Case 2.0 ebook, http://www.ivarjacobson.com/Use_Case2.0_ebook

Estimation of probabilistic worst case execution time while accounting OS costs

W. Talaboulma*, C. Maxim*,+, A. Gogonel*, Y. Sorel* and L. Cucu-Grosjean*
*INRIA Paris-Rocquencourt
Le Chesnay, France,
firstname.lastname@inria.fr
+Airbus SAS
Toulouse, France
firstname.lastname@airbus.com

***Abstract*—The arrival of modern and more complex processors (e.g., use of caches, multi- and many-core processors) increases the timing variability of tasks, i.e., the worst case execution time (WCET) is becoming significantly larger, while the probability of appearance of a worst case execution time is extremely low. Approaches taking into account this probability are based on the definition of the probabilistic worst case execution time of a task as a probabilistic bound on all execution time scenarios. Existing measurement-based approaches consider the execution of the tasks in isolation and our contribution provides hints for accounting OS costs.**

I. Introduction and related work

During the last twenty years different solutions have been proposed to time critical system designers through a pessimistic estimation of performances of the processors (thus increased costs) while using average time behavior processors. For instance DARPA estimates that in 2050 the construction of an airplane with current solutions will require the entire defense budget of USA [1].

The arrival of modern and more complex processors (e.g., use of caches, multi- and many-core processors) increases the timing variability of tasks, i.e., the absolute worst case execution time is becoming significantly larger, while the probability of appearance of a worst case execution time is extremely low [4]. Approaches taking into account this probability have been proposed either by measurement-based reasoning [8] [9] [13] [6] or static reasoning [2] [3]. The preemption costs are studied in presence of randomized caches [7] or simple processors [12]. Another stream of work considers that the worst case execution time of a task obtained in isolation is not becoming larger in the presence of the operating system (OSs) [10]. Such operating system is called compositional. In our work we are interested in studying the estimation of WCET of a task for an OS that is not compositional.

Our contribution Our paper concerns the probabilistic estimation of the WCET for non-compositional OSs. We consider the case of independent tasks scheduled using an off-line scheduler on one processor. We use measurement-based approaches for the estimation of the WCET to guarantee an approach that is scalable on more complex processors. **This paper presents the first measurement-based approach considering OS costs.**

II. Probabilistic worst case execution time (pWCET)

Definition 1. *The probabilistic execution time (pET) of the job of a task describes the probability that the execution time of the job is equal to a given value.*

For instance the j^{th} job of a task τ_i may have a pET

$$\mathcal{C}_i^j = \begin{pmatrix} 2 & 3 & 5 & 6 & 105 \\ 0.7 & 0.2 & 0.05 & 0.04 & 0.01 \end{pmatrix} \quad (1)$$

If $f_{\mathcal{C}_i^j}(2) = 0.7$, then the execution time of the j^{th} job of τ_i has a probability of 0.7 to be equal to 2.

The definition of the probabilistic worst-case execution time (pWCET) of a task is based on the relation $\succeq$ between two probability distributions, provided in Definition 2.

Definition 2. *[11] Let $\mathcal{X}$ and $\mathcal{Y}$ be two random variables. We say that $\mathcal{X}$ is worse than $\mathcal{Y}$ if $F_{\mathcal{X}}(x) \leq F_{\mathcal{Y}}(x)$, $\forall x$, and denote it by $\mathcal{X} \succeq \mathcal{Y}$. Here $F_{\mathcal{X}}$ is the cumulative distribution function of $\mathcal{X}$.*

In Figure 1 $F_{\mathcal{X}_1}(x)$ never goes below $F_{\mathcal{X}_2}(x)$, meaning that $\mathcal{X}_2 \succeq \mathcal{X}_1$. Note that $\mathcal{X}_2$ and $\mathcal{X}_3$ are not comparable.

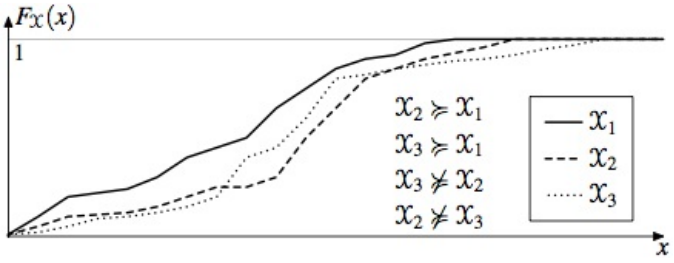


Fig. 1. Possible relations between the CDFs of various random variables

Definition 3. *The probabilistic worst case execution time (pWCET) $\mathcal{C}_i$ of a task τ_i is an upper bound on the pETs $\mathcal{C}_i^j$ of all jobs of τ_i $\forall j$ and it may be described by the relation $\succeq$ as $\mathcal{C}_i \succeq \mathcal{C}_i^j$, $\forall j$.*

Graphically this means that the CDF of $\mathcal{C}_i$ stays under the CDF of $\mathcal{C}_i^j$, $\forall j$. Thus the **probabilistic worst case execution time is upper bounding all probabilistic execution times** of a task.

III. Measurement-based of the pWCET of a task

A measurement-based approach for estimating the pWCET of a task has two main parts: (i) collecting the execution time traces of the task; and (ii) estimation the pWCET based on the set of execution time traces obtained during the first step.

The second step is mainly today done by using the Extreme Value Theory [8] [9] [13] [6]. This theory provides a pWCET estimation from a set of tasks by fitting the data to the closest Gumbel probability distribution. The Gumbel probability distribution is a particular case of the Generalised Extreme Value distribution which has the following cumulative distribution function:

$$F_\xi(x) = \begin{cases} e^{-(1+\xi\frac{x-\mu}{\sigma})^{\frac{1}{\xi}}} & \xi \neq 0 \\ e^{-e^{-\frac{x-\mu}{\sigma}}} & \xi = 0 \end{cases}$$

This distribution is defined by three parameters: shape (ξ), scale (σ) and location (μ). If the shape parameter $\xi = 0$ then F_ξ is a Gumbel distribution.

Existing work considers the execution time traces obtained in isolation. Our contribution is obtaining the execution time traces while different instances of the task are executed in presence of an OS, whose preemption cost is accounted. For this purpose we propose a methodology for answering the first step (i) of a measurement-bases approach.

IV. Estimation of pWCET while accounting OS costs

This paper is an initial step forward solving the more general problem consisting in estimating the WCET of a task while OS costs are accounted. We consider here the same framework as the one defined in [12] (see page 720, Section VIII). The tasks are executed on the same processor. The scheduler policy is defined with respect to an off-line static table.

During the execution of the tasks, we consider two cases: (a) the preemption costs are contained within the WCET estimation as an exact absolute worst case value and (b) the preemption cost is not approximated within the WCET and we measure its possible different values.

For our simulations we consider the interval defined by $(0, S_i + P)$, where

- $S_1 = O_1$;
- $S_i = \max\{O_i, O_i + \lceil \frac{S_{i-1}-O_i}{T_i} \rceil T_i\}, \forall i \in \{2, 3, \ldots, n\}$.

where P is the lcm of all periods and O_i is the offset of task τ_i [5].

The steps of our collection of execution times traces is done within the following steps:

STEP 1 The task is executed in isolation on the processor and a pWCET estimate is obtained using a method similar to [6].

STEP 2 The absolute worst case preemption cost is calculated as the maximum number of preemptions from all higher priority tasks.

STEP 3 The system is executed on the processor within the feasibility interval $(0, S_n + P)$, where n is the number of tasks.

We obtain two pWCET estimates: one solving problem (a) and a second solving a problem (b). The two estimates may be compared using a function like $CRPS = \sum_{i=0}^{+\infty}\{f_{pWCET_1}(i) - f_{pWCET_2}(i)\}^2$. This allows to calculate the pessimism of absolute worst case preemption costs with respect to the more finer approach that takes into account the variation of preemption costs during the execution.

V. Conclusion

We provide in this short paper a description of our solution for taking into account OS costs in WCET estimation of a task, without a good knowledge of the internal structure of the processor. Thus our method has the advantage of being scalable to more complex processor. We are currently obtaining the first numerical results after implementing a basic off-line static table. A first extension of our work is the introduction of more complex processors like caches or pipelines and of course more complex classes of schedulers.

References

[1] **. Augustine's laws. *the 6th edition of AIAA*, 1997.

[2] S. Altmeyer, L Cucu-Grosjean, and R. Davis. Static probabilistic timing analysis for real-time systems using random replacement caches. *Real-Time Systems*, 51(1):77–123, 2015.

[3] S. Altmeyer and R. Davis. On the correctness, optimality and precision of static probabilistic timing analysis. In *Design, Automation & Test in Europe Conference & Exhibition*, pages 1–6, 2014.

[4] F. J. Cazorla, E. Quiñones, T. Vardanega, L. Cucu, B. Triquet, G. Bernat, E. D. Berger, J. Abella, F. Wartel, M. Houston, L. Santinelli, L. Kosmidis, C. Lo, and D. Maxim. Proartis: Probabilistically analyzable real-time systems. *ACM Trans. Embedded Comput. Syst.*, 12(2s):94–114, 2013.

[5] L. Cucu-Grosjean and J. Goossens. Exact schedulability tests for real-time scheduling of periodic tasks on unrelated multiprocessor platforms. *Journal of Systems Architecture - Embedded Systems Design*, 57(5):561–569, 2011.

[6] L. Cucu-Grosjean, L. Santinelli, M. Houston, C. Lo, T. Vardanega, L. Kosmidis, J. Abella, E. Mezzeti, E. Quinones, and F.J. Cazorla. Measurement-based probabilistic timing analysis for multi-path programs. In *the 24th Euromicro Conference on Real-time Systems*, 2012.

[7] R. Davis, L. Santinelli, S. Altmeyer, C. Maiza, and L. Cucu-Grosjean. Analysis of probabilistic cache related pre-emption delays. In *25th Euromicro Conference on Real-Time Systems*.

[8] S. Edgar and A. Burns. Statistical analysis of WCET for scheduling. In *the 22nd IEEE Real-Time Systems Symposium*, 2001.

[9] J. Hansen, S Hissam, and G. A. Moreno. Statistical-based wcet estimation and validation. In *the 9th International Workshop on Worst-Case Execution Time (WCET) Analysis*, 2009.

[10] L. Kosmidis, E. Quiñones, J. Abella, T. Vardanega, and F. Cazorla. Achieving timing composability with measurement-based probabilistic timing analysis. In *16th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*, 2013.

[11] J. López, J. Díaz, J. Entrialgo, and D. García. Stochastic analysis of real-time systems under preemptive priority-driven scheduling. *Real-Time Systems*, pages 180–207, 2008.

[12] F. Ndoye and Y. Sorel. Monoprocessor real-time scheduling of data dependent tasks with exact preemption cost for embedded systems. In *the 16th IEEE International Conference on Computational Science and Engieering*.

[13] L. Yue, I. Bate, T. Nolte, and L. Cucu-Grosjean. A new way about using statistical analysis of worst-case execution times. *ACM SIGBED Review*, September 2011.

Tunable Response Time Upper Bound for Fixed Priority Real-Time Systems

Qiong Lu, Albert M. K. Cheng
Department of Computer Science
University of Houston
Houston, TX 77204, USA
cheng@cs.uh.edu

Robert I. Davis
Department of Computer Science
University of York
York, YO10 5GH, UK
rob.davis@york.ac.uk

ABSTRACT: *In exact response time analysis, determining the interference from higher priority tasks requires the ceiling operation, which is time-consuming. Enhanced Audsley's Algorithm (EAA) divides the set of tasks into two groups and uses different forms to indicate the time contribution, improves the efficiency. An important problem is how to separate the tasks. This paper presents a method to demarcate the tasks of higher priority. We first split these tasks by the approximation Fisher and Baruah have defined, which introduced a parameter k, and formalized the precision-tunable exact response time algorithm. Furthermore, we develop this algorithm into a response time upper bound with precision-adjustability.*

1. MOTIVATION

The exact-test algorithm for checking the schedulability of a task set achieves its precision by a time-consuming recurrence process. To overcome this computation complexity, sufficient-schedulability-test algorithms approximate tasks' utilization or response time via linear or polynomial bound. This paper introduces an approach to either improve the effectiveness of these sufficient tests or the efficiency of the exact test.

2. HYBRID ALGORITHM

Based on the work of Fisher and Baruah [1] and Nguyen et al [5], we derive a response time upper bound for fixed-priority real-time tasks more efficiently using the new Hybrid Algorithm with the following key steps. Assume each task is characterized by a series of parameters: its worst-case computation time (C_i), relative deadline (D_i), and period (T_i).

A. Dividing the tasks of higher priority: each iteration divides these tasks into two subsets *H* and *L*. If $(k-1)*T_j >= Prep$, task G_j goes to subset *L*; otherwise, task G_j goes to subset *H*, where *k* is predefined precision parameter (a positive integer), and T_j and *Prep* are respectively the period of the task of higher priority and the response time derived in the previous iteration.

B. Calculating the interference time: subset *H* is consuming time according to its tasks' utilization. The extra upper bound parts $C_j*(1-U_j)$ is also computed for these tasks, where U_j is task T_j's. The rest of the time remains for subset *L* and current task. The tasks of higher priority in subset *L* will employ a ceiling operation to calculate the interference time.

C. Finding the next response time: the worst-case response time formula is modified as follows:

$$R_i^{n+1}{}_{ub\_hybrid} = \frac{C + \sum_{\forall j \in hp(i) \cap j \in L} \left\lfloor \frac{R + J_j + T_j - 1}{T_j} \right\rfloor * C_j + \sum_{\forall j \in hp(i) \cap j \in H} C_j * (1 - U_j)}{1 - \sum_{\forall j \in hp(i) \cap j \in H} U_j} \quad (1)$$

Actually, it is same as formula (2) given in [4].

$$R_i^{n+1}{}_{ub\_hybrid} = \frac{C_i + \sum_{\forall j \in hp(i) \cap j \in L} \left\lceil \frac{R_i}{T_j} \right\rceil * C_j + \sum_{\forall j \in hp(i) \cap j \in H} C_j * (1 - U_j)}{1 - \sum_{\forall j \in hp(i) \cap j \in H} U_j} \quad (2)$$

3. RELATED RESEARCH

The model for the exact response time test is defined as

$$w_i^{x+1} = C_i + \sum_{j \in hp(i)} C_j * \left\lceil \frac{w_i^x}{T_j} \right\rceil \quad (3)$$

The worst-case response time can be derived by the following recurrence relation: the series of W_i^x, W_i^0 W_i^1 ... is a monotonically non-decreasing array where $W^{x+1}{}_i = W^X{}_i$; thus the worst-case response time can be derived.

Enhanced Audsley's Algorithm (EAA) of Lu et al [3] reduced the overhead of algorithm of the Exact Response Time algorithm by increasing response time. The main idea is elucidated as follows: the tasks of higher priorities are further split into two subsets according to a varying threshold value in each iteration. One subset called *H* consists of the tasks with higher priorities. This group of tasks consumes time proportional to their utilization, leaving time for lower priority subset called *L* and current task T_i.

Fisher and Baruah [1] have developed a polynomial-time approximation scheme for schedulability analysis in static priority systems. Then Fisher et al [2] consider jitter and assume the all the task parameters are integers, extending this formula, which has a tighter upper bound and improves the approximation function. As a result, the upper bound parts are changed into $(t+T-1)*U$. Nguyen et al [5] improve this by

*Supported in part by the US National Science Foundation under Awards No. 0720856 and No. 1219082.

assuming that all parameters are not necessarily integer and a tighter bound can be derived. The upper bound is $(t+T-C)*U$.

4. Illustration of the Hybrid Algorithm

Consider this example. Given a task set $S= \{S_1= (C_1=2, T_1=4\}, S_2= \{C_2=1, T_2=5\}, S_3= \{C_3=3.3, T_3=15\}\}$, the schedulability of S_3 is evaluated by the hybrid method. Suppose that both S_1 and S_2 are schedulable and for a precision parameter k, we set k as 2 and 10, respectively. We set C_3=3.3 as the initial response time for S_1 because r =3.3 is less than (2-1)*4. The computation of interference time from S_1 uses the ceiling operation. For S_2, due the same reason, r =3.3 is less than (2-1)*5 so that it also uses the ceiling operation to calculate the interference time. Therefore, the next response time R_{next}=3.3+2+1=6.3. S_3 becomes schedulable when k is equal to 10 rather than 2. Meanwhile, the number of the iterations also increases from 2 to 6. Consequently, as k increases, the precision of the Hybrid Algorithm improves but incurs more time overhead. As a result, the number of tasks "mistakenly" classified as unschedulable due to the lower precision could be redressed as schedulable as the precision parameter k increases. As R_3=14.3 is less than both $(k-1)*T_1$ and $(k-1)*T_2$, the test actually has become the exact test for S_3 with k=10.

5. Empirical Evaluation

This experiment investigates the effectiveness of the approach. The overall utilization varies for task sets from *0.65* to *1* in increments of *0.05*, and the precision parameter k changes from 1 to 10. The other parameters use default values: $N=24$, $M=3$, $D_i=T_i$, $B_i=J_i=0$, *1000* task sets are generated to calculate the percentage of task sets that are schedulable for each utilization level.

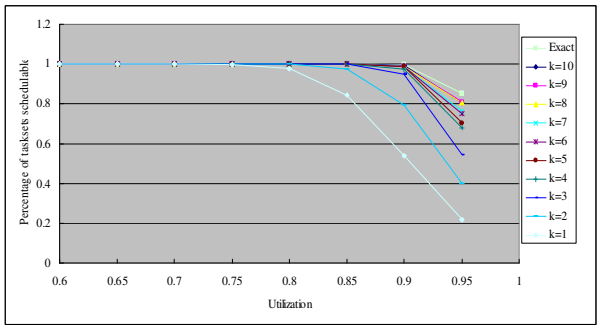


Figure 1: Percentage of schedulable task sets with $D=T$ versus utilization. Data is for the Hybrid and Exact Response Time Algorithms.

As shown in **Figure 1**, as the precision parameter k increases, the percentage of schedulable task sets increases and converges to the exact test. Fewer number of higher-priority tasks are consuming time according to the utilization so that the response time relatively increases. Consequently, the schedulability relatively improves if other parameters remain the same. In addition, the curves are getting closer together as the response time increases more slowly, with the result that the response time gap between different k's for each task becomes smaller as k increases.

Essentially, the Hybrid algorithm is a sufficient schedulability analysis approach. Therefore, it is necessary to compare its effectiveness with other sufficient algorithms. **Figure 2** indicates that the Hybrid algorithm is not only more effective than other sufficient algorithms, but it also becomes accurate as k increases. Additionally, the Hybrid Algorithm also dominates the "Response Time Upper Bound" algorithm.

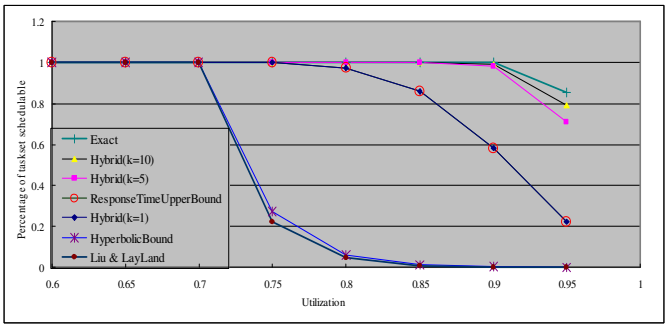


Figure 2: Percentage of schedulable task sets with $D=T$ versus utilization. Data is for the Hybrid Algorithm, Exact Response Time Algorithm, Response Time Upper Bound, Hyperbolic Bound and Liu and Layland Bound.

The majority of the task sets are schedulable at *95%* of utilization or below according to the exact test. By comparison, using the Hybrid Algorithm ($k=5$ and $k=10$), the majority of the task sets are also deemed schedulable at *95%* utilization or below. For the Response Time Upper Bound, most of the tasks are deemed schedulable at *90%* utilization or below. It is worth noting that the Hybrid algorithm actually becomes the Response Time Upper Bound Algorithm when k is set to 1. As shown in **Figure 1**, the curve for the Hybrid Algorithm with k equal to 1 overlaps with the curve for the Response Time Upper Bound. This experiment shows that the Hybrid Algorithm is a precision tunable and effective schedulability analysis algorithm. Our ongoing work performs more analysis and experiments, and the results will be reported in a full paper.

References

[1] N. Fisher and S. Baruah. A Polynomial-Time Approximation-Scheme for Feasibility Analysis in Static-Priority Systems with Bounded Relative Deadline, Proc. 17th Euromicro Conference on Real-Time systems, pp. 117-126, July 2005.

[2] N. Fisher, T. H. C. Nguyen, J. Goossens, and P Richard. Parametric Polynomial-Time Algorithms for Computing Response-Time Bounds for Static-Priority Tasks with Release Jitters, Proc. 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), 2007.

[3] W.-C. Lu, J.-W. Hsieh, W.-K. Shih and T.-W. Kuo. A Faster Exact Schedulability Analysis for Fixed Priority Scheduling, J. Systems and Software, Vol. 79, No. 12, pp.1744-1753, Dec.2006.

[4] Qiong Lu, Master's Research Report, University of York, 2009.

[5] T. H. C. Nguyen, P. Richard, and E. Bini, Approximation Techniques for Response-Time Analysis of Static-Priority Tasks, J. Real-Time Systems, Vol. 43, pp. 147–176, 2009.

Model-Predictive Controllers for Performance Management of Composable Conveyor Systems

Shunxing Bao, Aniruddha Gokhale
EECS, Vanderbilt Univ
Nashville, TN, USA
Email: {shunxing.bao,a.gokhale}@vanderbilt.edu

Sherif Abdelwahed
ECE, Mississippi State Univ
Starkville, MS, USA
Email: sherif@ece.msstate.edu

Shivakumar Sastry
ECE, Univ of Akron
Akron, OH, USA
Email: ssastry@akron.edu

***Abstract*—Composable Conveyors expose fundamental new problems that must be addressed as the nation transforms its advanced manufacturing infrastructure. Unanticipated fluctuations in workloads caused by the increasingly open and interconnected advanced manufacturing systems makes it significantly challenging to appropriately configure and adapt the operating parameters of conveyor systems that are deployed at individual plants such that reliability and desired quality-of-service (QoS) requirements are met. Moreover, these must not be tightly coupled to a single layout but work seamlessly after the conveyor layouts change. To address such challenges, this paper describes different controller design strategies and compares each approach against a baseline system without any controller.**

***Index Terms*—Composable conveyors, controller design**

I. Introduction

Material handling and packaging systems are excellent examples of widely-used engineered systems that embody many characteristics of cyber physical systems (CPS). Such systems have applications in warehouses, manufacturing plants, package sorting facilities (e.g., FedEx and UPS), and even in frontline logistics for military deployments. Composable conveyor systems (CCS) provide composable and reconfigurability in the layout of an assembly or material handling plant that makes them attractive in application scenarios because they can adapt to changing process and environmental demands – an emerging need for advanced manufacturing systems that aspire seamless interconnection across the supply chain and the plant floor [1].

CCS, like other automation systems, are becoming increasingly open and more connected to the supply chain. Workloads in the system can fluctuate substantially; e.g., holiday season may experience dramatic increase in packages that must be sorted and shipped to their destinations. In such situations, statically-defined preset speeds for the entities of the CCS may not be sufficient to effectively operate the system. Similarly, unforeseen disruptions in the supply chain can make any fixed schedule ineffective. Plant operators are thus faced with the task of addressing at least two key challenges.

- First, they must be able to dynamically adapt the operation of their plant to maximize the throughput and minimize energy consumption while adapting to the unanicipated workload fluctuations.
- Second, these dynamic adaptation capabilities must remain available when the conveyor topology or layout of the plant floor undergoes change due to business specific and other logistical reasons.

Our prior work for CCS has explored the use of model driven engineering tools to reason about different properties of CCS conveyor layouts prior to their actual deployment [2]. More recently we adapted the classical priority inheritance protocol to resolve priority inversions in CCS [3]. None of these efforts investigated the use of model predictive control. On the other hand, we have designed a model-predictive, two-level controller for the adaptive performance management of computing systems, however, this solution was applied to a purely cyber-only system [4]. In the current work we focus on cyber physical systems and account for both the physical dynamics and the cyber interactions of these sysetms. To that end we are exploring three alternative model predictive approaches to controller design for CCS. The rest of this paper provides the status of our ongoing work.

II. Alternate Designs of Model Predictive Controllers for CCS

A. Controller Design

System Model: The composable conveyor systems (CCS) we consider for our research comprises multiple instances of two kinds of units, namely, Segment and Turn. A Segment has a belt whose speed and direction can be controlled. A Turn is a reconfigurable merger/splitter unit that can be juxtaposed with upto four Segment instances. These conveyor systems can be controlled by regulating the speed of the belt on the individual Segment units. The load in the system, i.e., the number of packages handled by different units in the system can be regulated by dynamically routing the packages over different end-to-end paths in the system. Configuring and synchronizing the speeds of the Segments is unlikely to be scalable if the decisions are made in a central location. The speeds must also be adjusted in response to variabilities in the arrival rates of the packages at the different inputs to the system.

Our current work focuses on addressing the first challenge, i.e., predict unforeseen workload and autonomically configure the minimum belt speed needed for energy conservation and maintaining the maximum throughput. In our design, we allow six speed levels for the belts: two levels for low, two for medium, and two for high. For our ongoing work, the conveyor

topology is fixed. Figure 1 and the logistics are predefined. I1 and I2 denote input bins, S1,S2, ..., S8 represent segment belts; T1, ..., T4 are switch turns; and O1 and O2 are output bins. Two flows are fixed for the incoming workloads according to the arrow directions shown in Figure 1

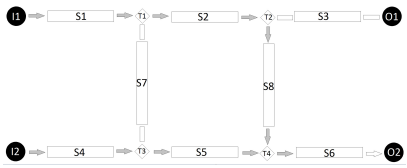


Fig. 1. MainTopology

We use an ARIMA model to make the estimation and adjust the parameter based on the patterns of arrival. The local controller is a limited look-ahead controller (LLC). We also define a cost function to measure the QoS. This function takes into account the cost of the control inputs themselves and their change. It is also possible to consider transient costs as part of the operating requirements, expressing the fact that certain trajectories towards the desired state are preferred over others in terms of their cost or utility to the system.

In short, the LLC controller predicts for the next time step the package arrival rate, traverse all 6 possible speed values (in two-level controller, it has a difference that is explained later), and calculate cost function, and finally choose the speed which can make the next cost function minimum and achieve energy saving. Energy consumption is directly related to belt speed. The following four strategies are designed:

(1) No controller (NC): The systems units (conveyor belts) have fixed speed. The system uses maximum speed to make sure no package is dropped.

(2) Totally decentralized controllers (TD): Each unit has a local LLC controller and predicts arrival rates independently of each other. This approach is easy to extend but lacks precision and incurs unstability with large variability in arrival rate.

(3) Partially decentralized controllers (PD): Each unit has a local LLC controller but it predicts the arrival rate taking into account the speed of the preceding unit. This is more accurate than the previous approach but requires additional communication.

(4) Two-level global controller (GC): This approach is based on a completely decentralized controller design, where the individual controllers at the first level make short-term predictions of workloads. These controllers are in turn managed by a second level controller that makes longer-term forecasts of expected workloads and fine tunes the performance of the system. The global controller decides for the next 60 time units the speed level so that the system can only traverse limited times of speed level. The two level controller can reduce the space complexity from $O(n^2) \rightarrow O(n)$.

B. Preliminary Results

We present preliminary results of simulation experiments we performed. To compare the strategies, we apply the same topology and the same package arrival rate at the input bins.

Prediction evaluation: Figure 2 presents prediction behavior of a segment S1 in our topology using a ARIMA model. The green line is estimation error, and the average estimation accuracy rate is 88.13%. If the package workload changes dramatically, we observe larger error differences because incoming package arriving rate is sharply different from previous workload.

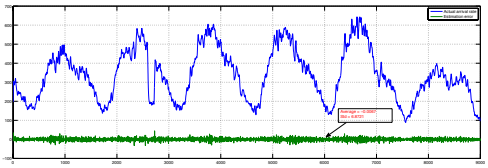


Fig. 2. Belt S1's Prediction Evaluation in TD

Energy Conservation: Table I summarizes the energy conserved by the three strategies when compared against the no controller strategy.

TABLE I
ENERGY CONSERVATION EVALUATION

Simulation	Energy Conservation%
TD	83.62
PD	83.95
GC	74.39

III. CONCLUDING REMARKS

Our current work focuses on developing controllers for autonomic performance management of composable conveyor systems used in advanced manufacturing. This work presented preliminary results on performance and energy savings accrued using three different model predictive controller designs. Our ongoing work is evaluating the designs on different conveyor topologies and workload arrivals based on real-world workloads. All simulations are available for download from https://github.com/onealbao/Composable_Predictable_Controller

ACKNOWLEDGMENTS

This work has been supported in part by NSF CNS CAREER award 0845789.

REFERENCES

[1] AMP 2.0 Steering Committee, "Accelerating U.S. Advanced Manufacturing," President's Council of Advisors on Science and Technology (PCAST), Tech. Rep., Oct. 2014.

[2] K. An, A. Trewyn, A. Gokhale, and S. Sastry, "Model-driven Performance Analysis of Reconfigurable Conveyor Systems used in Material Handling Applications," in *Second IEEE/ACM International Conference on Cyber Physical Systems (ICCPS 2011)*. Chicago, IL, USA: IEEE, Apr. 2011, pp. 141–150.

[3] S. Sastry and A. Gokhale, "Resolving Priority Inversions in Composable Conveyor Systems," *Elsevier Journal of Systems Architecture (JSA)*, vol. 60, no. 6, pp. 509–518, Jun. 2014.

[4] N. Kandasamy, M. Khandekar, and S. Abdelwahed, "A Hierarchical Optimization Framework for Autonomic Performance Management of Distributed Computing Systems," in *Proceedings of the 26th IEEE International Conference on Distributed Computing Systems (ICDCS 06)*, Lisboa, Portugal, Jul. 2006.

Towards Certifiable Multicore-based Platforms for Avionics

M. Ali Awan*, P. Meumeu Yomsi*, K. Bletsas*, V. Nélis*, E. Tovar*, P. Souto†
*CISTER/INESC-TEC, ISEP, Polytechnic Institute of Porto, Portugal
{muaan, pamyo, ksbs, nelis, emt}@isep.ipp.pt
†University of Porto, FEUP-Faculty of Engineering
pfs@fe.up.pt

Motivation and Challenges

The demand for extra functionality in modern applications is a never ending trend. The hardware vendors are actively improving the design of processors to accommodate these complex applications. The increase in clock speed to enhance the performance of the processor has hit its limits. This is driven by the fact that the performance per watt became costly at high frequencies. Hence, Moore's law is no longer sustained with increasing frequencies but with additional cores [1]. Therefore, in the last decade, the semiconductor industry has experienced a paradigm shift from single processor design to multicore processors (MCP). Cores in MCP share many resources like caches, main memory, I/O devices and interconnects. This sharing, which does not exist in single core processors, makes the temporal behavior of MCPs rather complex and highly unpredictable as these platforms are designed to improve the average-case performance. Consequently, their use in safety-critical applications such as avionics domain is extremely challenging. The certification authorities are very skeptical in the use of MCP platforms in avionics applications.

In May 2014, the North and South American aviation authorities together with their European and Asian counterparts, more specifically the "Certifications Authorities Software Team (CAST)" published a position paper CAST-32 [2] where they express their concerns w.r.t. the use of two-core processors in the implementation of safety-critical avionics systems. Therein, they identify different sources of non-determinism in modern MCPs and suggest guidelines to overcome them.

In order to secure the certifiability of MCPs for avionics, we believe there are three different ways to handle these sources of non-determinism: 1) developing a predictable MCP hardware; 2) providing an extra hardware support to existing MCPs with FPGA(s) (field programmable gate arrays) to circumvent non-deterministic paths and 3) propose software-based mechanisms to mitigate the effect of non-determinism in current MCPs. Although there is a strong industrial drive towards developing hardware-based solutions to the problem, we also firmly believe on the merits of software-based mechanisms being used to mitigate the non-determinism arising from resource sharing in currently available MCPs. Therefore in this paper we focus on some of those software-based mechanisms. In particular, we are exploring timing models that would: (i) accurately incorporate the cache related preemption and migration delays (CRPMD), (ii) mitigate the non-deterministic effect of interconnects, and (iii) consider the interference caused by I/O subsystem on memory, interconnect and caches.

This work was partially supported by National Funds through FCT/MEC (Portuguese Foundation for Science and Technology) and when applicable, co-financed by ERDF (European Regional Development Fund) under the PT2020 Partnership, within project UID/CEC/04234/2013 (CISTER Research Centre); also by FCT/MEC and the EU ARTEMIS JU within project(s) ARTEMIS/0001/2013 - JU grant nr.621429 (EMC2) and ARTEMIS/0003/2012 - JU grant nr.333053 (CONCERTO); and also by FCT/MEC and ERDF through COMPETE (Operational Programme 'Thematic Factors of Competitiveness'), within project FCOMP-01-0124-FEDER-020447 (REGAIN).

I. CRPMD

Assuming a preemptive and migrative scheduler and MCPs, a premier source of unpredictability stems from sharing caches. This results in an increase in the WCETs of the tasks, unfortunately. We plan to circumvent this issue as follows. First, we strongly believe that by reasoning about the subset of potentially conflicting cache blocks for every pair of preempting/preempted tasks, and bounding the number of preemptions and migrations suffered by each task, we can derive tighter bounds on the WCET for every given scheduling policy. Hence, we opt for such a fully analytical approach for CRPMD estimation [3]–[5]. We will consider two different scheduling approaches: NPSF [6] and C=D [7]. These schedulers are chosen for their ability to achieve high utilization bounds. Upon deriving sound analysis, tweaks to the scheduling algorithms will be needed so that they perform well. One idea to hybridize them is to allow some tasks to migrate only at the job boundary to mitigate the number of migrations. This approach has two advantages: (1) It simplifies the analysis on the one hand and (2) reduces the pessimism on the other. Another idea with potential involves identifying opportunities for deviation from the standard policy at runtime, when it would be safe and provable to reduce CRPMD overheads. Delaying some preemptions/migrations or swapping the task execution order are some ideas in that direction. Finally, on a more practical front, we plan to implement all of this in a real system. Although it is not an architecture for safety critical systems, we plan to target x86Linux for two reasons: to leverage our existing native implementations based on NPSF and C=D and also as a proof of concept. If these techniques work for x86Linux multiprocessors, then we can be confident that the same will apply to predictable platforms.

II. Interconnects

In MCP system-on-chip, the interconnect is a key component. Its timing behavior impacts virtually any operation at the architectural level. Some efforts have been devoted to derive a timing model for this component in the avionics domain. However, most of the available techniques are based on assumptions which are not met in modern MCPs. Our goal is to develop a model of the CoreNet as used in Freescale's P4080 for example, and more recently, in its P5020 multicores. A risk that can prevent us from reaching this objective may be the non-disclosure of essential information by Freescale [8]. However, we were told by sources in Airbus [9] that Freescale has recently been more forthcoming in providing implementation details. Hence, we expect to have access to the necessary information. As a fall-back plan, we will analyze a wormhole Network-on-Chip (NoC) using virtual channels. NoC is the de facto technology that is meant to become mainstream in future multi/manycore processor chips, because it has been proven through extensive experiments over the last years that any bus topology performs poorly when the number of cores exceeds 8 [10]. Thus any bus-based technology is not a scalable solution. Independently of the interconnect used, we plan to validate the model developed experimentally. To this end, we will use embedded systems benchmarks that are known to be representative of the kinds of applications in avionics [11]. In addition, we will try to obtain from Airbus and Embraer [12] a characterization of novel avionics applications. The novelty of the contribution in this section will be to shift and put the focus on the practicality of the solution proposed. This will inherently make our work different from most works in the state of the art as we will give more importance to the simplicity, practicality, and safety of the solution rather than its efficiency.

III. I/O subsystem

Most safety-critical applications in avionics require the I/O devices to interact with the environment on one front and perform some functions on the other. Among aforementioned resources, I/O devices are also shared among cores on MCPs. The generated traffic from different shared resources (cache, memory, I/O, core) may interfere with each other. For example, CoreNet Coherency Fabric in Freescale P4080 platform connects cores with last-level cache, memory and I/O devices. One of the major issues in such MCP platforms is to develop timing models to tackle the effect of traffic generated from all shared resources. We intend to explore different aspects of the I/O subsystem interference on various shared resources. More precisely, we will address the following two issues.

a) I/O and traffic generated from cores: Most of the peripherals are based on buffered data. In this case, the requests received over a period of time are buffered before initiating the direct memory access (DMA) transfer from the peripheral to the main memory. In this context, a request may not suffer the same delay as the previous one. On the other hand, every core requests to fetch data from the main memory on a cache miss. The size of the requests is usually few bytes (e.g., cache line size, typically 64bytes), so hardware prefetchers sometimes may combine requests for multiple cache lines. However, the size of the request is still small (perhaps two or three cache lines) when compared to a DMA transfer. On top of this, speed of the core is too high when compared to the main memory. Hence, the pipeline is likely to stall when a core retrieves a cache line from main memory. So, the delay suffered by the requests made by a core is cumulative (each request delays the next one), while it is not necessarily the case for peripherals (if there is enough buffer). This should fundamentally change the way the analysis is performed. By considering the difference between buffered and non-buffered traffic, we can achieve better bound in the timing analysis.

b) I/O devices and caches: I/O devices also affect the cache traffic in several ways. For instance, some of the MCP platforms (such as Freescale P4080) allow the DMA units in the peripherals to write directly to cache instead of transferring data to main memory. On one side, this feature (called cache stashing) allows cores to directly read the data from the cache instead of generating any memory requests and loading the contents in the cache. On the other side, this mechanism complicates the cache analysis as designer needs information about the instructions of such transfers. There is a need to develop the timing analysis to cope with this issue by providing a mechanism to differentiate between the DMA transfer bounded to memory or cache. There is a possibility in some MCP platforms to configure some memory as either cache or scratchpad. I think that using scratchpad for allocating I/O data (which has to be managed through a driver, so we have some direct control over buffers and addresses) could be a good way to avoid some of the aforementioned problems (interference, coherency, etc.).

References

[1] D. Geer, "Chip makers turn to multicore processors," Computer, vol. 38, no. 5, pp. 11–13, 2005.

[2] "Certification authorities software team (cast), position paper (cast-32) multicore processors," Certification authorities in North and South America, Europe, and Asia, May 2014.

[3] W. Lunniss, S. Altmeyer, G. Lipari, and R. I. Davis, "Accounting for cache related pre-emption delays in hierarchical scheduling," in 22st RTNS, ser. RTNS '14. ACM, 2014, pp. 183–192.

[4] W. Lunniss, R. I. Davis, C. Maiza, and S. Altmeyer, "Integrating cache related pre-emption delay analysis into edf scheduling," in 19th RTAS, ser. RTAS '13. IEEE Computer Society, 2013, pp. 75–84.

[5] S. Altmeyer, R. Davis, and C. Maiza, "Cache related pre-emption delay aware response time analysis for fixed priority pre-emptive systems," in 32nd RTSS, Nov 2011, pp. 261–271.

[6] K. Bletsas and B. Andersson, "Preemption-light multiprocessor scheduling of sporadic tasks with high utilisation bound," in 30th RTSS, Dec 2009, pp. 447–456.

[7] A. Burns, R. Davis, P. Wang, and F. Zhang, "Partitioned edf scheduling for multiprocessors using a c=d task splitting scheme," J. Real–Time Syst., vol. 48, no. 1, pp. 3–33, 2012.

[8] F. Semiconductor. [Online]. Available: http://www.freescale.com

[9] A. c. Airbus. [Online]. Available: http://http://www.airbus.com/

[10] B. Nikolic and S. M. Petters, "Real-time application mapping for many-cores using a limited migrative model," J. Real–Time Syst., 1 2015.

[11] J. Nowotsch, M. Paulitsch, D. Buhler, H. Theiling, S. Wegener, and M. Schmidt, "Multi-core interference-sensitive wcet analysis leveraging runtime resource capacity enforcement," in 26th ECRTS, July 2014, pp. 109–118.

[12] E.-E. B. de Aeronautica. [Online]. Available: http://www.embraer.com