ECRTS

7th-10th July 2015 *Lund, Sweden*

Work-in-Progress Proceedings

27th Euromicro Conference on Real-Time Systems
July 7 – 10, 2015
Lund, Sweden

Edited by Harini Ramaprasad

Message from the Work-in-Progress Chair

Dear Colleagues:

Welcome to Lund, and to the Work-in-Progress (WiP) Session of the 27th Euromicro Conference on Real-Time Systems (ECRTS'15). This session is dedicated to promising new and ongoing research on real-time systems and applications. I am happy to present six excellent WiP papers that cover innovative research from a wide range of topics, including real-time scheduling, timing analysis, real-time issues on multi-core architectures and avionics platforms, and real-time databases. I am confident that many of the research contributions we feature here will appear as full-fledged conference and journal papers in the near future. The proceedings will be published online via the ECRTS 2015 WiP session website: http://control.lth.se/ecrts2015/wip.html.

The primary purpose of the WiP session is to provide researchers with an opportunity to discuss their evolving ideas and to gather feedback from the real-time community at large. Due to time constraints, the presentations in this session can only provide a brief overview of the new creative ideas and interesting approaches of the selected research contributions. Nevertheless, I hope that you all will enjoy this session, and that you will find the ideas presented interesting. Most of all, I hope you will participate in stimulating discussions, exchange your ideas, and provide valuable feedback to the authors.

I would like to thank the members of the WiP session technical program committee for their timely help in reviewing the papers. I would also like to thank the authors for their interesting contributions and their confidence in ECRTS as a means to improve and advance their research. Last but not least, special thanks go to the ECRTS'15 organizers, Karl-Erik Årzén, Steve Goddard, and Gerhard Fohler, for their support.

Harini Ramaprasad
University of North Carolina at Chalotte

Work-in-Progress Chair
ECRTS 2015

ECRTS '15 Work-in-Progress Technical Program Committee

Program Committee

- Cong Liu, University of Texas at Dallas, USA
- Ahlem Mifdaoui, University of Toulouse/ISAE, France
- Sibin Mohan, University of Illinois at Urbana Champaign, USA
- Vincent Nelis, CISTER/ISEP, Portugal
- Gabriel Parmer, George Washington University, USA

Work-in-Progress Session Chair

Harini Ramaprasad, University of North Carolina at Charlotte, USA

Table of Contents

Towards Scalable Configuration of Time-Division Multiplexed Resources

Anna Minaeva, Přemysl Šůcha, Benny Akesson, Zdeněk Hanzálek
Czech Technical University in Prague

I. Introduction

The trend of consumer-electronics systems becoming more and more complex is not possible to overlook. The number of cores, reflecting the complexity of such systems, has been growing exponentially and is expected to continue this trend in the coming decade [1]. However, due to the typically short life-cycle of these systems, the design time is required to remain unchanged. Therefore, the design of such systems is pushed to be automated.

Systems with a number of applications are considered, where some applications can have *firm real-time requirements* and must always satisfy their deadlines, while for other non-real-time applications, sufficient average performance is required. Cores, executing the applications are resource *clients*. They share resources, such as memories, buses and peripherals. Resource sharing causes *contention* that must be resolved by an *arbiter*. Time-Division Multiplexing (TDM) is a commonly used arbiter for many types of resources.

This work deals with automated configuration of TDM arbiters. Such a configuration requires an approach that constructs a schedule that efficiently *satisfies the bandwidth and latency requirements* of the clients, while *minimizing their resource utilization* (maximizing slack capacity) to improve the average performance of non-real-time clients. This schedule is an assignment of time slots to the clients. In this configuration process, it is crucial that the approach is *scalable*, i.e. that it is able to find a solution in reasonable time to problems with a large number of clients in order to satisfy the needs of contemporary and future consumer-electronics systems.

The three main contributions of this work are: 1) We present a schematic description of an optimal approach that takes an existing integer linear programming (ILP) model addressing the TDM configuration problem and wrap it in the branch-and-price framework [2] to improve the scalability, 2) We present a stand-alone heuristic approach that can be used to find the slot allocation, providing a trade-off between computation time and efficiency 3) We experimentally evaluate the scalability of the branch-and-price approach, comparing both to the previously formulated ILP model and to an existing heuristic.

A. Related Work

Most works in the considered field of consumer-electronics systems design that use exact optimization techniques do not scale well enough to be able to handle the complexity of future systems [3]–[5], while only a few try to come up with advanced techniques to cope with this issue. These techniques can be classified into two major groups of approaches: 1) a decomposition of the problem into smaller sub-problems, and 2) navigating the search smartly during design-space exploration. The first approach deals with large problems by solving many smaller problems instead, trading the exponential growth of complexity for a polynomial one. This method is used in [6], [7]. The second branch of improvements uses problem-specific information while searching the design space, which is profitable comparing to using general design-space exploration methods. Authors in [8] and [9] use boolean satisfiability and ILP approaches, respectively, while looking for a minimal reason of constraint violation and trying to prevent this situation in the rest of the search.

To the best of our knowledge, this work is the first to apply an advanced optimization approach, called branch-and-price [2], in the field of consumer-electronics systems design. Branch-and-price aims to combine both of the mentioned improvements, i.e. it decomposes the problem into smaller sub-problems and it allows using more sophisticated search-space exploration methods.

B. Background

Relevant background, necessary for a reader to understand the main contributions of the paper is presented below. Latency-rate ($\mathcal{LR}$) [10] servers is a shared resource abstraction that guarantees a client sharing a resource a minimum allocated rate (bandwidth), ρ, after a maximum service latency (interference), Θ, as shown in Figure 1. The figure illustrates a client requesting service from a shared resource over time (red line) and the resource providing service (blue line). The $\mathcal{LR}$ service guarantee, the dashed line indicated as service bound in the figure, provides a lower bound on the amount of data that can be transferred to a client during any interval of time when the client is actively requesting service. A formal definition of a $\mathcal{LR}$ server is provided in Definition 1.

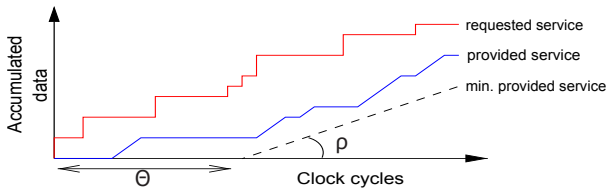


Fig. 1. A $\mathcal{LR}$ server and associated concepts for a client sharing a resource.

Definition 1: A server is a $\mathcal{LR}$ server if and only if a non-negative service latency Θ_i can be found such that the provided service, r_i^j, of a client i is bounded by Equation (1) during a time interval of duration j when the client is actively requesting service. The minimum non-negative constant Θ_i satisfying Equation (1) is the service latency of the server.

$$r_i^j \geq \max(0, \rho_i \cdot (j - \Theta_i)) \quad (1)$$

The way values of Θ and ρ of each client are obtained depends on the particular choice of an arbiter and how it is configured. The TDM arbitration belongs to the class of $\mathcal{LR}$

servers and it operates by periodically repeating a schedule with a fixed number of slots, *f*, each corresponding to a single resource access with bounded execution time in clock cycles. While the bandwidth requirement determines the minimum number of slots allocated to a client, the service latency requirement put constraints on the assignment of allocated slots in the schedule according to Definition 1. Determining service latency Θ for an arbitrary slot assignment has an $O(f^2)$ complexity.

II. Problem Formulation

The problem of finding a TDM slot allocation with a given frame size that satisfies the requirements of a *set of clients*, while minimizing the rate allocated to real-time clients is given here. An instance of the considered problem is defined by a tuple of requirements $\langle C, \hat{\Theta}, \hat{\rho}, f \rangle$, where $C = \{1, ..., n\}$ is the set of real-time clients that share a resource with n the number of clients; $\hat{\Theta} = [\hat{\Theta}_1, \hat{\Theta}_2, ..., \hat{\Theta}_n]$ and $\hat{\rho} = [\hat{\rho}_1, \hat{\rho}_2, ..., \hat{\rho}_n]$ are given *service latency* and *rate (bandwidth) requirements* of the clients, respectively, and *f* is a given TDM frame size.

A TDM schedule and its associated parameters are formalized by the set $F = \{1, 2, \cdots, f\}$, denoting TDM slots, while the sets $\Theta = [\Theta_1, \Theta_2, ..., \Theta_n]$ and $\rho = [\rho_1, \rho_2, ..., \rho_n]$ are the *service latency* and *allocated rate*, respectively, provided by the TDM schedule.

The goal is to find a schedule for n clients sharing the resource such that the objective function, Φ is minimized as shown in Equation (2), while the service latency and rate constraints (Equations (3) and (4)) are fulfilled.

$$\textit{Minimize:} \sum_{i \in C} \rho_i = \Phi \tag{2}$$

$$\rho_i \geq \hat{\rho}_i,\ i \in C \tag{3}$$

$$\Theta_i \leq \hat{\Theta}_i,\ i \in C \tag{4}$$

III. Branch-and-Price Approach

This section starts by introducing the existing ILP model in order to understand the proposed branch-and-price technique. The ILP model is a straightforward formulation of the considered problem, where binary decision variables x_i^j are introduced with $x_i^j = 1$ meaning slot j is allocated to client i and $x_i^j = 0$ otherwise. The details of this formulation can be found in [11].

To expand scale of the problems that we are able to solve by the ILP model, a branch-and-price method [2] is introduced. Branch-and-price allows solving instances of the considered problem with a larger number of clients, where the ILP formulation becomes too slow. The reasons are that by decomposing the problem it manages to reduce the problem size significantly. Moreover, it reduces the number of explored symmetrical solutions and typically has smaller branching tree. Both of these properties result in a significantly reduced computation time compared to the ILP model for large problem instances.

Branch-and-price is an exact method to solve ILP problems. In order to obtain a problem formulation for the branch-and-price approach, Dantzig-Wolfe decomposition [2] is performed on the ILP model. At the higher level, this decomposition transforms the space of binary variables x_i^j of the ILP model into the space of complete solutions for individual clients, i.e. branch-and-price works with complete schedules for individual clients, called columns, instead of dealing with single slot allocations.

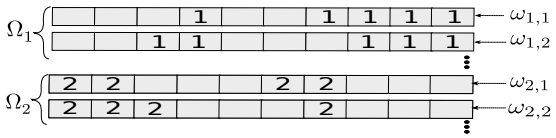


Fig. 2. Example of a set of columns.

A result of Dantzig-Wolfe decomposition on the ILP model is an ILP *master model* $MM(\Omega)$ that contains a *set of all possible columns for all clients* $\Omega = \{\Omega_1, \Omega_2, \cdots, \Omega_n\}$. Columns are iteratively generated by a so called *sub-model*, here the ILP model, for a single client. Then they are combined into a complete solution for all clients by the master model. The formulations of the master model and sub-model are left out of this paper due to the page limit.

An example column set is shown in Figure 2. In the considered problem, columns are TDM schedules for individual clients. Here, the set of columns Ω_1 for the first client contains columns on top of the figure and the set of columns Ω_2 for the second client is at the bottom. The decision variables $\omega_{i,p}$ of the master model indicate whether or not column p is included in the schedule for client i. One of the possible solutions here is to use column 2 for the first client and column 1 for the second one, i.e. $\omega_{1,1} = 0$, $\omega_{1,2} = 1$, $\omega_{2,1} = 1$, $\omega_{2,2} = 0$.

A drawback of using columns instead of binary variables of the ILP model is the large number of possible columns. However, it is sufficient to gradually generate only the most promising ones, since adding these columns to the master model gradually extends the search space of solutions. At a certain (final) moment it can be proven (see [2]) that the optimal solution is found. A master model that considers only a subset of columns $\Omega^R \subseteq \Omega$ is called *the Restricted Master Model* and is denoted by $MM(\Omega^R)$.

Thus, the idea, described above, is known as the *column generation* approach. Since column generation is only able to solve the linear relaxation of the master model, it is necessary to extend this approach with a *branch-and-bound* technique in order to be able to get an integer solution. This combination is known in the literature as *branch-and-price*.

A. Outline of the algorithm

The overall scheme of the branch-and-price algorithm is shown in Figure 3. First of all, the algorithm must assign initial columns to Ω^R in Step 1, which is done by a heuristic, e.g. from Section IV. Step 2 starts the process of column generation by solving the linear relaxation of the restricted master model, which means that the obtained solution $\omega_{i,j}$ is not integral in general. The output of this step is quantitative directions (dual values) that guide the column search for the sub-model. Next, a new column for some client is constructed by the sub-model (Step 3), considering the directions obtained in the previous step by $MM(\Omega^R)$. If a new promising column is found for any client, the column is added to Ω^R and the next iteration of the column generation algorithm starts. Otherwise, the optimal solution of the relaxed $MM(\Omega^R)$ is *a new lower bound to the optimal integral solution* and is denoted by Φ^{LB}. If the bounding takes place in Step 4, i.e. this branch is already

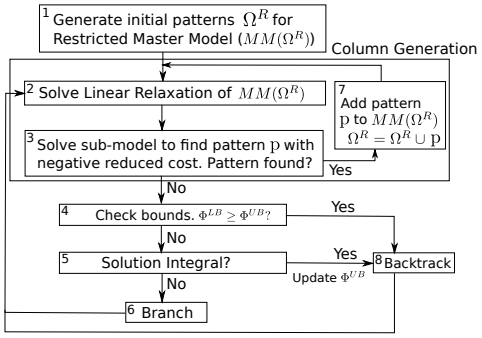


Fig. 3. Outline of the branch-and-price algorithm.

worse or the same as the best solution known so far (upper bound on the criterion), the current node is closed in Step 8. In case the branch is still promising, Step 5 checks whether or not the solution, obtained by column generation, is integral, i.e. the values $\omega_{i,j}$ are integral and the final schedule is found. In case it is, a new candidate solution to the initial integer master model is found. This solution defines a new *upper bound on the criterion* Φ^{UB}, which is updated and the node is closed in Step 8. In case neither bounding nor check on integrality close the node, branching takes place.

Using the branch-and-bound technique means having a branching tree, which is in essence a set of nodes with parent-child relationships. The node is defined by a partial solution, i.e. a chain of slot assignment decisions made in the parent nodes. The column generation procedure (Steps 2, 3, 7) together with bounding (Step 5) and checking solution on integrality (Step 4) are launched in each node. In case the node is not closed, a child node is generated with a new decision made. A decision is to assign/forbid assignment of a slot to a client. The algorithm is finished when the whole branching tree is explored in such a manner.

IV. Heuristic

Although the proposed exact approach solves the considered problem optimally, sometimes it is desirable to have a faster solution of a lower quality. Furthermore, branch-and-price can use a heuristic solution as a good starting point. The proposed heuristic exploits the ILP model from [11] for a single client, which is the sub-model from the branch-and-price approach in Section III. Iteratively running the sub-model for different clients in a round robin manner, the complete schedule is constructed. The sub-model aims to minimize $\sum_{j \in F} x_{i,j} \cdot \lambda_j$, where the coefficients λ_j in the column generation procedure are set by the master model in Step 2 in Figure 3. In the heuristic Steps 2 and 7 of Figure 3 are substituted by Algorithm 2 that heuristically assigns appropriate coefficients λ_j.

Algorithm 1 shows the scheme of the proposed heuristic. The inputs are the number of clients n, service latency Θ_i and bandwidth ρ_i requirements for each client. Furthermore, there are two parameters of the heuristic: a coefficient α, which controls the speed of convergence to the final solution, and the maximum number of iterations (sub-model runs) N_{iter}^{max}. These parameters allow a trade-off between computation time and quality of the solution. Each iteration includes two main steps: coefficients λ_j are computed on Line 4 and then the sub-model for client i is launched on Line 5.

Algorithm 1 Heuristic

1: Inputs: $n, \Theta, \rho, \alpha, N_{iter}^{max}$
2: $N_{iter} = 0$, $i = 1$, $x_{i,j}^{curr}$ (current schedule)
3: **while** $N_{iter} < N_{iter}^{max}$ and $x_{i,j}^{curr}$ has collisions **do**
4: λ = ComputeCoefficients(c_i, α)
5: $x_{i,j}^{curr}$ = SubModel($\Theta_i, \rho_i, \lambda$)
6: $N_{iter} = N_{iter} + 1$
7: $i = (i \mod n) + 1$
8: **end while**
9: Output: $x_{i,j}^{curr}$

The core of the heuristic is the assignment of coefficients λ_j to each slot for a given client such that in case another client or clients try to allocate the same slot, some of the clients will change their allocation. The procedure of assigning coefficients λ_j for client i is presented in Algorithm 2, where in case a) client i is not allocated in the considered slot, but there is a client $k \neq i$ that has this slot allocated in the current schedule. Then the coefficient is set to allow allocation of this slot by the client i on the earlier stages and to prohibit it in the later stages of the run depending on how many times this slot was allocated before in the schedule; b) client i is the only one using this slot in the current schedule, the coefficient is constant for the client to prefer allocation of this slot; c) there is a conflict in slot j and client i is a part of the conflict. Since it is not clear in advance which client should have the slot, randomness is introduced here; d) no client has allocated slot j in the current schedule, the coefficient is set to be constant, but slightly more than in case b) to prefer not to change the allocation when it is not necessary.

Algorithm 2 ComputeCoefficients

1: Inputs: $i; \alpha$;
2: **for all** $j \in 1, \cdots, f$ **do**
3: $d_{j,i}$ = number of times slot j was allocated before to any client $k \neq i$
4:
$$\lambda_j = \begin{cases} min(2, 1 + d_{j,i} \cdot \alpha), & x_{k,j}^{curr} = 1,\ k \neq i. \quad \text{(a)} \\ 0.9, & \text{if } x_{i,j}^{curr} = 1,\ x_{k,j}^{curr} = 0, \\ & \forall k \neq i. \text{ (b)} \\ 1 + rand() \cdot 1.5, & \text{if } x_{i,j}^{curr} = 1,\ x_{k,j}^{curr} = 1, \\ & k \neq i. \text{ (c)} \\ 1, & \text{if } x_{k,j}^{curr} = 0,\ \forall k \in C. \text{ (d)} \end{cases}$$
5: **end for**
6: Output: λ

V. Preliminary Results and Conclusion

The experiments evaluate the scalability of the proposed branch-and-price approach and the trade-off between computation time and the total rate (the criterion) allocated for $200 \cdot 4$ use-cases with 8, 16, 32 and 64 real-time clients with the frame size of 64, 128, 256 and 512, respectively, for the optimal and heuristic approaches. Moreover, it compares the proposed approaches with an already existing exact (ILP formulation) strategy, mentioned in Section III. In order to show both the advantages of the branch-and-price and the heuristic approach, the use-cases are synthetically generated so that both their bandwidth and latency requirements are approximately equally demanding in terms of the number of allocated slots.

The generation process first randomly assigns bandwidth requirement ρ_i to each client i from a defined interval of [0.06, 0.14], [0.03, 0.07], [0.015, 0.035] and [0.0075, 0.0175] for 8, 16, 32 and 64 clients, respectively. The bandwidth requirements of the use-case are accepted if the total required rate of all clients is in the range [0.7, 0.9]. Otherwise, the use-case is discarded and the generation process restarts. Service latency requirements Θ are uniformly distributed according to $\frac{1}{\gamma \cdot \rho}$, where a larger value of γ indicates a tighter requirement and the value γ is randomly chosen from the [0.95, 1.4], [0.9, 1.3], [0.85, 1.2], [0.8, 1.1] for 8, 16, 32 and 64 clients, respectively. These numbers are calibrated to give approximately the same total allocation of all use-cases, making it possible to compare them in terms of computation time.

A time limit of 3 000 seconds per use-case was set in order to obtain the results of the experiments in a reasonable time. Furthermore, due to the complexity and desirable trade-off the heuristic was launched 8 times for each use-case in order to improve the results. The parameters are set as $\alpha = 0.1$ and $N_{iter}^{max} = 250$. Notice that since the heuristic contains randomness its result can change in different runs.

Figure 4 shows log_{10} of the computation time for the heuristic, branch-and-price and ILP approaches for 8, 16, 32 and 64 clients. We conclude that *the existing ILP model does not scale to use-cases with 32 and 64 clients*. Out of 11 use-cases with 32 clients the ILP model was able to find a feasible solution only in 3 use-cases with the given time limit and the average computation time for the use-case is around 2500 seconds, thus it is not reasonable to run all 200 of them. Thus, the results for 32 and 64 clients are represented by the heuristic on the left and branch-and-price on the right. For the use-cases with 8 clients, branch-and-price and the ILP model behave similarly with slight advantage to the ILP formulation. The branch-and-price approach is able to solve all the use-cases in 6 hours, while the ILP model needs only 3 hours. Such a difference is a result of branch-and-price not beeing able to prove the optimality for 3 use-cases (and run for 3000 seconds each), while the ILP failed only once. For 16 clients the ILP model runs 20 hours, while the branch-and-price approach needs less than 3 hours, resulting in 4.4 times time reduction. Moreover, branch-and-price solve all the use-cases to optimality, but ILP failed in 1 use-case, where it found the optimal solution, but was not able to prove its optimality with the given time limit.

The heuristic fails to find a feasible solution in 23 and 26 use-cases for 8 and 16 clients, respectively, but saves 95% and 88% of the computation time of the fastest optimal approach. The heuristic shows good results, especially on the use-cases with 32 and 64 clients, where it is able to solve almost all of them (fails in 8 for 32 and 0 for 64 clients) in 90 minutes and 16 hours, respectively. The solution obtained by the heuristic lies in less than 0.01% from the solution obtained by branch-and-price on average. Thus, the heuristic is fast and gives near-optimal results.

From this experiment, we confirm the exponential complexity of the problem, although our implementation solves instances with 64 clients and 512 slots in less than 5 minutes on average for the 200 use-cases. Moreover, the ILP model is not able to solve use-cases in reasonable time, starting approximately from 32 clients. Thus, the branch-and-price approach is able to improve the scalability, solving larger problem instances, while ILP shows better results for the use-cases with smaller number of clients. Furthermore, the proposed heuristic enables to save up to 95% of the computation time with maximally 13% of not feasible solutions, where the distance from the found to the optimal solution does not exceed 1%.

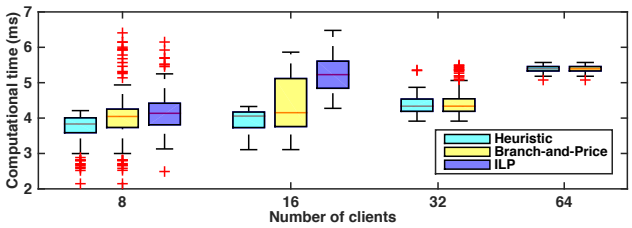


Fig. 4. Computation time distribution for the use-cases.

The plans for the future work are to improve the performance of the branch-and-price approach and to conduct more experiments, including more use-cases of different nature - those that are more bandwidth demanding and those that are more service latency demanding.

ACKNOWLEDGEMENT

This work was supported by the Grant Agency of the Czech Republic under the Project GACR P103/12/1994, the Ministry of Education of the Czech Republic under project number CZ.1.07/2.3.00/30.0034, and Eaton European Innovation Centre.

REFERENCES

[1] "International Technology Roadmap for Semiconductors (ITRS)," 2011.
[2] D. Feillet, "A tutorial on column generation and branch-and-price for vehicle routing problems," *4OR*, p. 1, 2010.
[3] M. D. Gomony *et al.*, "Architecture and Optimal Configuration of a Real-Time Multi-Channel Memory Controller," in *Proc. DATE*, 2013.
[4] Y. Yi, W. Han, X. Zhao, A. Erdogan, and T. Arslan, "An ilp formulation for task mapping and scheduling on multi-core architectures," *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09*, pp. 33–38, April 2009.
[5] J. Lin, A. Gerstlauer, and B. Evans, "Communication-aware heterogeneous multiprocessor mapping for real-time streaming systems," *Journal of Signal Processing Systems*, vol. 69, no. 3, pp. 279–291, 2012.
[6] S. Wildermann, M. Glaß, and J. Teich, "Multi-objective distributed run-time resource management for many-cores," in *Proceedings of the Conference on Design, Automation & Test in Europe*, ser. DATE '14. 3001 Leuven, Belgium, Belgium: European Design and Automation Association, 2014, pp. 221:1–221:6. [Online]. Available: http://dl.acm.org/citation.cfm?id=2616606.2616877
[7] W. Liu, M. Yuan, X. He, Z. Gu, and X. Liu, "Efficient sat-based mapping and scheduling of homogeneous synchronous dataflow graphs for throughput optimization," *Real-Time Systems Symposium, 2008*, pp. 492–504, Nov 2008.
[8] F. Reimann, M. Lukasiewycz, M. Glass, C. Haubelt, and J. Teich, "Symbolic system synthesis in the presence of stringent real-time constraints," in *Proceedings of the 48th Design Automation Conference*, ser. DAC '11. New York, NY, USA: ACM, 2011, pp. 393–398. [Online]. Available: http://doi.acm.org/10.1145/2024724.2024817
[9] M. Lukasiewycz and S. Chakraborty, "Concurrent architecture and schedule optimization of time-triggered automotive systems," in *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*. ACM, 2012, pp. 383–392.
[10] D. Stiliadis and A. Varma, "Latency-rate servers: a general model for analysis of traffic scheduling algorithms," *IEEE/ACM Trans. Netw.*, vol. 6, no. 5, 1998.
[11] B. Akesson, A. Minaeva, P. Šůcha, A. Nelson, and Z. Hanzálek, "Efficient configuration methodologies for time-division multiplexed resources," *RTAS*, 2015.

Some Results in Rate Monotonic Scheduling with Priority Promotion

Mitra Nasri and Gerhard Fohler
Chair of Real-time Systems,
Technische Universität Kaiserslautern, Germany
{nasri,fohler}@eit.uni-kl.de

***Abstract*—Rate monotonic (RM) scheduling algorithm cannot guarantee schedulability of highly utilized tasks in all cases. In this paper, we increase schedulability of RM by assigning fixed set of priority promotions for each task. We present an algorithm to assign promotion times (PT) and the new priority at each PT. Each priority promotion occurs after a constant relative time from the release of an instance of a task. This technique can be implemented using a container at the operating system's side which sets a timer interrupt for the priority promotion. Our priority promotion algorithm is offline and it promotes priorities only at the latest release of the higher priority tasks before deadline of the first instance of each task. In our solution, the maximum number of promotions is bounded to the number of higher priority tasks with smaller periods. The performance of our solution approach has been evaluated by extensive sets of simulations. According to our experimental results, a) our solution dominates RM, i.e., it guarantees schedulability whenever the task set is RM-feasible while it improves the schedulability in the task sets which are not RM-feasible, b) it has a significantly higher schedulablity ratio than RM, and c) even in cases where it cannot guarantee the schedulability, it has a low ratio of missed jobs.**

I. Introduction

It is known that for task sets with implicit deadline, the rate monotonic (RM) scheduling algorithm is optimal among other fixed-priority (FP) algorithms [1]. However, RM guarantees 100% utilization only in special cases, e.g., in harmonic task sets [2]. Unlike dynamic priority-based scheduling algorithms such as EDF [1], RM has considerably lower run-time overhead, however, it cannot guarantee schedulability in general cases of periodic tasks with high utilization.

The idea of *priority promotion* is introduced in [3] along with the concept of dual-priority tasks where each task is allowed to promote its priority at most once. The system uses a fixed-priority scheduling algorithm. The priority promotion happens after a specified amount of time from the release of each job of the task. In [4], open problems for the existence of a feasible dual-priority assignment has been discussed. It have been shown that for the special case of systems with two tasks, there exists such an assignment which guarantees schedulability up to 100% utilization. Recently, in [5], an upper bound on the priority promotions per job has obtained for EDF. This bound is a function of the largest deadline in the task set. It has been proven that the worst-case response time (WCRT) of tasks with priority promotions scheduled by RM will not necessary happen in their first release, which means that the *critical instant* is no longer at the first release of the tasks. It inherently increases the complexity of finding a feasible priority promotion solution. In [5], no algorithm has been presented to promote priority of the tasks.

In this paper, we propose a method to increase RM schedulability by assigning a fixed set of priority promotions to each task. The priority promotions happen at a fixed relative time from the *release* of each job. For each promotion time (PT), we define a new priority which is higher than the last priority of the task before the latest PT. The basic idea of the algorithm is to set priority promotions for the first release of the low priority task at the times at which high priority tasks have their latest release before the deadline of the low priority task. Using these promotion times, RM and EDF behave almost the same, at least for the first instance of the tasks. Since the maximum number of promotions for each task is limited to the number of higher priority tasks, it is bounded to $n-1$, where n is the number of the tasks. The algorithm itself has $O(n^2 log(n))$ computational complexity.

Priority promotion times and the new priorities are assigned at design-time. To be able to apply them at run-time, each task has a container, implemented on the operating system's level, which governs promotion times (using a timer interrupt). Doing so provides separation of concerns for the application's designers and maintains security at the system level such that the applications cannot modify their own priorities.

The remainder of the paper is organized as follows; the system model is presented in Sect. II. In Sect. III, we introduce our priority promotion algorithm. Efficiency of the algorithm is evaluated in Sect. IV in terms of schedulability ratio and job miss ratio. Some discussions about the algorithm and its schedulability analysis has been provided in Sect. V, and finally, the paper is concluded in Sect. VI.

II. System Model and Problem Statement

We assume a uni-processor system and a hard real-time task set $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ with n independent periodic tasks. Each task τ_i is described by $\tau_i : (c_i, T_i)$, where c_i is the worst-case execution time and T_i is the period of the task. We assume synchronous task sets with no release offsets. Tasks have implicit deadlines, i.e., for each task, the deadline is equal to the period. They are indexed such that $T_1 \leq T_2 \leq \ldots \leq T_n$. The utilization of the task set is denoted by $U = \sum_{i=1}^{n} u_i$ where $u_i = c_i/T_i$. Moreover, the hyperperiod of this task set is the least common multiplier (LCM) of the periods, and is denoted by H.

For each task, we assign a set of time instants at which the priority is promoted. The j^{th} priority promotion of task

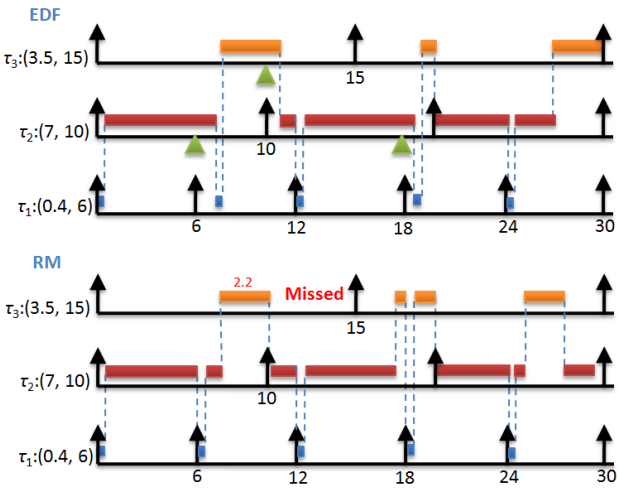


Fig. 1. EDF schedule versus RM schedule in a task set with 100% utilization

τ_i is identified by $P_{i,j} : (\delta_{i,j}, p_{i,j})$ for $1 \leq j \leq m_i$ where $0 \leq \delta_{i,j} < T_i$ is the priority promotion time, $p_{i,j}$ is the new priority indicator, and m_i is the number of priority promotions of task τ_i. Since in RM, the priority indicator is the period of the task, i.e., the smaller the period, the higher the priority, we assume $p_{i,j}$ contains a value which is smaller than the current task's period and it can be used as the promoted priority for the current instant of the task.

III. Priority Promotion Approach

In this section we present a simple idea to derive effective instants of time, i.e., PT, where priority promotion helps RM to behave more like EDF. We introduce our priority promotion assignment algorithm (PPA) which will be used at design-time to find constant relative time instants (w.r.t. the release of each job of the task) at which the priority can be promoted.

For two tasks τ_i and τ_j, $T_j < T_i$, RM priorities and EDF priorities are exactly the same for the jobs of τ_j which have deadlines before the deadline of current instant of τ_i. For those jobs, both EDF and RM prioritize τ_j over τ_i. The only difference happens for the latest job of τ_j which is released before deadline of τ_i and its deadline is after deadline of τ_i. Fig. 1 shows this situation for a task set with 3 tasks at their first releases. As shown in this figure, if we promote priority of τ_3 at 10 at least for the first instance of τ_3, EDF and RM schedule become identical.

Since in a synchronous task set, the first instance of each task encounters the highest workload, we assign priority promotions according to the first instance of the tasks. Our priority promotion assignment algorithm finds the latest releases of the tasks with smaller deadlines than τ_i. Those releases are denoted by r_g for $1 \leq g < i$. In the first step we sort those release times and then, starting from the earliest release, we add a priority promotion to the list of promotions of τ_i. Parameters of this promotion are $\delta_{i,j} = r_g$ and $p_{i,j} = T_i - r_g$ where j is the index of the current priority promotion. Then all releases of the tasks with $T_q \geq T_g$ will be removed from the list because if τ_i gains a priority which is higher than

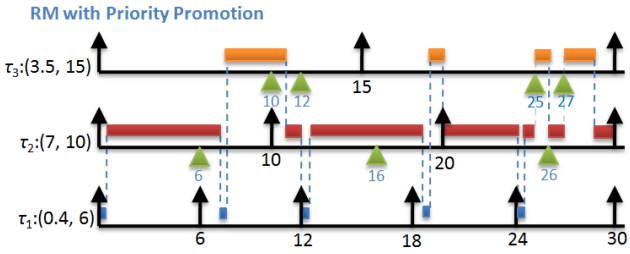


Fig. 2. RM with priority promotion schedule for the task set in Fig. 1

τ_g, it also has a higher priority than τ_q. After removing those items from the list of releases, we continue to the next earliest release and add the next priority promotion time until there is no other item in the list of releases. This algorithm has been shown in Alg. 1. Fig 2 shows the result of RM with the priority promotion for the previous example. Comparing this figure with Fig. 1 we see that these three algorithms produce different schedules.

Algorithm 1 Priority Promotion Assignment Algorithm

Input τ: where τ is the set of tasks
Output P: P is the set of priority promotions

1: $P_1 \leftarrow \emptyset$
2: **for** $i \leftarrow 2$ **to** n **do**
3: $R \leftarrow$ latest releases time of task τ_1 to τ_{i-1} before T_i
4: Sort R in ascending order
5: $P_i \leftarrow \emptyset$
6: **while** (R is not empty) **do**
7: $r^{min} \leftarrow$ the earliest release in R
8: $g \leftarrow$ index of the task which is released at r^{min}
9: $\delta \leftarrow r^{min}$
10: $p \leftarrow T_i - r^{min}$
11: $P_i \leftarrow P_i \cup \{(\delta, p)\}$
12: Remove any r_k, $1 \leq k < i$ with $T_k \geq T_g$ from R
13: **end while**
14: **end for**
15: **return** P

Offline computational complexity of Alg. 1 is $O(n^2 log(n))$ because in Line 3 we must sort R which contains the list of the latest releases, and at most has $n-1$ items. We can use a double linked list during the construction of R to have an efficient implementation. Every item in this list is attached to one of the release values in R and points to the next item in R which has a larger period than the current item. Using this list, Line 12 can be implemented in $O(n)$ while the amortized cost of the while-loop remains $O(1)$ per task, because each task is either used in priority promotions or removed by Line 12. Note that we have at most $n-1$ tasks in the while-loop. As a result, computational complexity of each iteration of the for-loop in Lines 2 to 14 is $O(nlog(n))$ due to the sort operation, and hence, the algorithm runs in $O(n^2 log(n))$.

To apply priority promotion at run-time, each task can have an OS-level container which is equipped with a timer event. Starting from the release of each job, the container sets the next timer interrupt for the next priority promotion time from the given list which is produced by Alg. 1. If the task is finished before the next timer interrupt, the interrupt can be

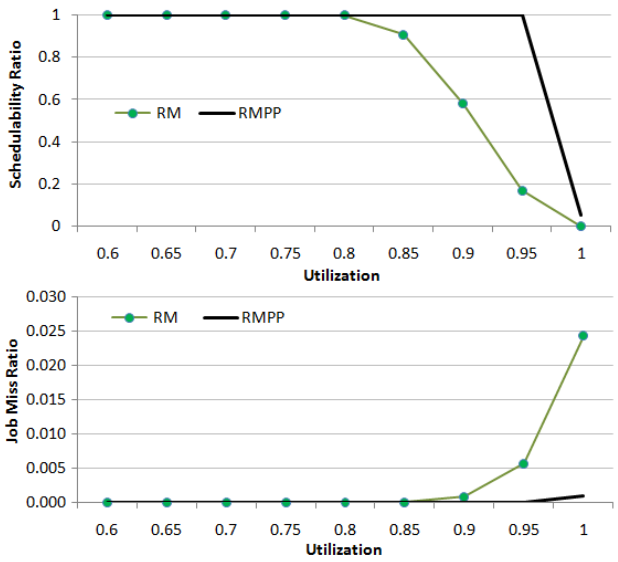


Fig. 3. Schedulability ratio and job miss ratio of the algorithms

canceled. In this implementation, permissions for promoting the priority belong to the operating system. Thus, there will be no issue regarding malicious promotions made by the applications themselves. More discussions about schedulability of our approach are presented in Sect. V.

IV. Experimental Results

In this section, we compare the performance of RM and RM with priority promotion (RMPP) which was described in Sect. III. Performance measures are job miss ratio and schedulability ratio where the former is the number of missed jobs divided by total number of jobs, and the latter is the number of schedulable task sets with no missed jobs divided by total number of generated task sets. Periodic task sets have been constructed according to [6] paper, i.e., in the first step, based on the given utilization U, n random u_i values have been obtained from uUniFast algorithm, and then, T_i has been selected randomly from $[10, 200]$ with uniform distribution. Based on u_i and T_i, c_i is calculated. Due to the space limitation, we have reported only the results for task sets with $n = 7$ tasks.

We use task set utilization U as the parameter of the experiment. It ranges from 0.1 to 1.0 with steps of 0.1. For each utilization value we have generated 200 random task sets as stated before. All resulting data is within a confidence interval ± 0.05 for confidence level 0.95. Fig. 3 shows the results of the experiment. Results of EDF algorithm have not been reported because it has schedulability 1 and miss ratio 0 in this setup.

In the next experiment, we only consider random task sets which are schedulable by RM. We have repeated the experiment for 10000 task sets per utilization value ranging from 0.6 to 1.0. Interestingly, we never found a case where RM guarantees schedulability while RMPP does not. In other words, based on our experiments, RMPP never misses a deadline if the task set is RM-feasible. Later in Sect. V we discuss

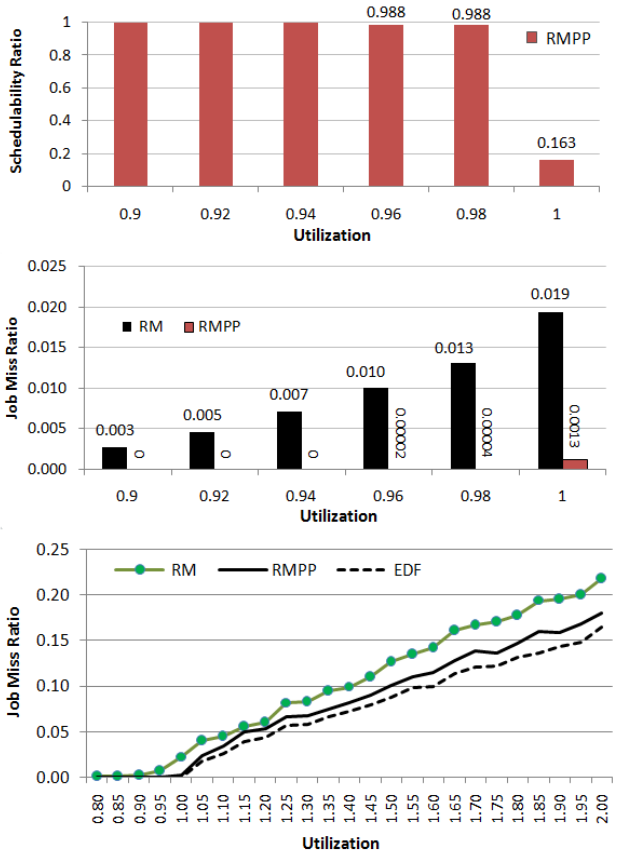


Fig. 4. Schedulability ratio and job miss ratio of RM and RMPP in the task sets which are not RM-feasible. Also for those task sets, we have compared job miss ratio of the algorithms with EDF in cases where $0.8 \leq U \leq 2.0$ which is an overloaded system

this observation with more detail. In the third experiment, we only consider random task sets which are not schedulable by RM algorithm. According to the results shown in Fig. 4, by adding fixed set of priority promotions assigned by Alg. 1 it is possible to efficiently increase RM schedulability. Even in the cases where hard real-time guarantee was not possible for RMPP, it has a very small miss ratio compared to RM.

V. Discussions

As shown by Fig. 3 and 4, our approach for promoting priorities is not optimal, i.e., it cannot guarantee all deadlines. Fig. 5 shows a case where the second instance of a task could not be scheduled even with the promoted priorities. The reason is that to assign promotion times for a task such as τ_i, Alg. 1 only considers set R of the latest releases which are constructed based on T_i. For the case of task sets with two tasks, [4] has shown that the optimal priority promotion assignment must consider the greatest common divisor of two periods. However, the problem remains unsolved for larger task sets with more than two tasks.

As mentioned in the second experiment of Sect. IV, according to our results, if a task set is RM-feasible, it is RMPP-

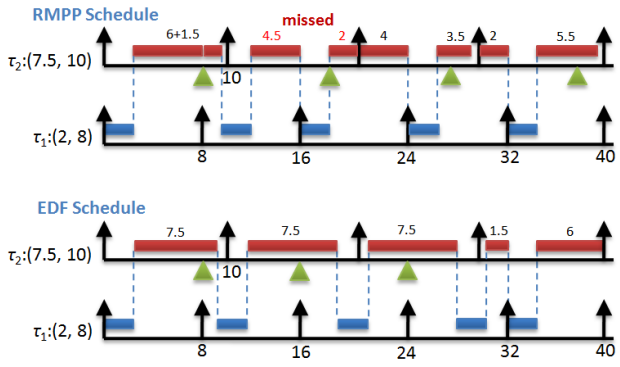


Fig. 5. A counter example to show non-optimality of PPA algorithm

feasible too. For justifying this observation we start with task sets with $WCRT_i \leq r^{min}$ for $1 \leq i \leq n$ where

$$r^{min} = \min \left\{ \left\lfloor \frac{T_i}{T_g} \right\rfloor T_g \right\}_{1 \leq g \leq i} \quad (1)$$

In these task sets, each task will be finished before it is promoted at r^{min} which is the first promotion time. Assume an RM-feasible task set is given which has $r^{min} < WCRT_i \leq r^{max} \leq T_i$ where r^{max} is the smallest release time among all release times in set R (produced in Line 3 of Alg. 1) which is larger than $WCRT_i$. If RM preserves schedulability, it means that within the schedule of higher priority tasks from time instant r^{min} to r^{max}, there will be enough idle time so that τ_i can finish its execution before its deadline. Thus, all higher priority tasks which are released between r^{min} and r^{max} must be finished before r^{max}, otherwise, RM will not start the remaining execution time of τ_i after r^{min}. Note that according to the definition, all of these tasks have deadlines greater than or equal to T_i, otherwise, it would not be their latest release before T_i. In an RMPP and EDF schedule, any of the jobs which are released from r^{min} to T_i will have lower priority than τ_i, consequently, they will be scheduled after τ_i. Due to the fact that the task set is RM-feasible, we know that all of the high priority tasks together with τ_i can be finished before r^{max}, which is smaller than or equal to T_i. Consequently, they can be finished before their deadline as well. As a result, at least for the first instance of the tasks in an RM-feasible task set, priority modifications of RMPP will not cause a deadline miss for any of the higher priority tasks. It is worth mentioning that priority modifications of any of the tasks with smaller period than τ_i has no effect on the schedule of τ_i because still all of those tasks have higher priority than τ_i.

The previous justifications are not complete because we did not consider future releases of the tasks. According to [5], in a fixed-priority algorithm with promoted priorities, the classic notion of critical instant will no longer be valid [5], i.e., the WCRT of the tasks may not appear in their first release. As a result, to provide a valid schedulability test or to prove the dominance of RMPP over RM, we need to consider future releases of the tasks too. As shown in Fig. 5, a deadline miss might happen for the second release of a low priority task because of the fact that in that release (or one of the future releases), promotions must happen earlier than the promotion times which we have assigned according to the first release of the task.

VI. Summary and Conclusion

In this paper we have used priority promotion approach with constant number of promotions per job to improve the schedulability of RM algorithm. Our solution, which is called RM with priority promotion (RMPP) has two phases; in the offline phase we assign promotion times and the new priorities respectively, and in the online phase, an OS-level container applies those priority promotions for each job of a task. Promotion times are obtained based on the latest release of the higher priority tasks under the first release of each task. This algorithm has $O(n^2 log(n))$ computational complexity. Moreover, the maximum number of promotions for each task is bounded to the number of higher priority tasks in the task set.

In our experimental results, our approach dominates RM, i.e., all RM-feasible task sets has been RMPP-feasible too. Besides, in task sets which were not RM-feasible, RMPP had a considerably high schedulability ratio. Also RMPP had a low miss ratio in cases where it cannot guarantee the schedulability. We have tried to justify the intuitive reasons behind the dominance of RMPP over RM, though, the actual proof remains as a future work. We will try to derive necessary or sufficient schedulability conditions for RMPP as well.

References

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

[2] C.-C. Han and H.-Y. Tyan, "A Better Polynomial-time Schedulability Test for Real-time Fixed-priority Scheduling Algorithms," in *IEEE Real-Time Systems Symposium (RTSS)*. IEEE Computer Society, 1997, pp. 36–45.

[3] A. Burns and A. J. Wellings, "Dual Priority Assignment: A Practical Method for Increasing Processor Utilisation," in *Euromicro Workshop on Real-Time Systems (EWRTS)*, 1993, pp. 48–55.

[4] A. Burns, "Dual Priority Scheduling: Is the Processor Utilisation bound 100% ," in *International Real-Time Scheduling Open Problems Seminar (RTSOPS)*, 2010, pp. 3–5.

[5] D. Masson, L. George, and J. Goossens, "Dual Priority and EDF: a closer look," in *Work-in-Progress Session of IEEE Real-Time Systems Symposium (WiP-RTSS)*, 2014.

[6] E. Bini and G. Buttazzo, "Measuring the Performance of Schedulability Tests," *Real-Time Systems*, vol. 30, no. 1-2, pp. 129–154, 2005.

Static CRPD-Aware Real-Time Scheduling

Guillaume Phavorin, Pascal Richard
LIAS, Université de Poitiers
Poitiers, France
{guillaume.phavorin,pascal.richard}@univ-poitiers.fr

Claire Maiza
Verimag, INP Grenoble
Grenoble, France
claire.maiza@imag.fr

***Abstract*—We consider the problem of scheduling hard real-time tasks subjected to cache-related preemption delays (CRPD) on uniprocessors. Most works in the literature focus on either improving the system predictability by bounding the CRPD or reducing conflicts between the tasks at the cache level. As the CRPD-aware scheduling problem is NP-hard even under simplified assumptions, we focus on finding a mathematical model for offline scheduling to compute an optimal solution to the CRPD-aware scheduling problem.**

I. Introduction

Nowadays, in order to reduce the costs, commercial off-the-shelf (COTS) components are more and more used in real-time embedded systems. Commercial processors now incorporate several micro-architectural features designed to improve the overall performances, such as cache memories. However, the use of cache memories in critical embedded systems raises a significant problem which is the question of predictability. Indeed, the execution time of an instruction depends whether this instruction can be found in the cache (cache hit) or has to be loaded from the main memory (cache miss), which can be up to 10 times more costly. Because multiple tasks have access to the cache during the system life, and as soon as preemptions are authorized, one task can overwritten some cache locations another task will needed when resuming its execution, causing additional reloads knows as Cache-Related Preemption Delays (CRPD).

To deal with such a problem, several solutions have been proposed in the literature. On one hand, some works, as in [1], focus on computing upper-bound on the CRPD to take them into account either in task worst-case execution times (WCET) or latter during the schedulability analysis. On the other hand, other works such as [2], have focused on reducing cache interference using partitioning or locking techniques. In both cases, no modification occurs at the scheduling level: classic algorithms such as Rate Monotonic (RM) or Earliest Deadline First (EDF) are still used. Some works as in [3] have focused on modifying scheduling algorithms to reduce the number of preemptions. But they do not consider any cache-related preemption delays. Very few works have considered the problem of explicitly taking optimal scheduling decisions based on CRPD, except for soft real-time multiprocessors as in [4].

In this paper we explicitly model *the* CRPD *as a timed penalty being an input for the scheduling algorithm*. Note that it is independent from any cache managing strategy, such as partitioning. CRPD values are assumed to be computed beforehand by a timing analysis. This CRPD-aware scheduling problem NP-hard has been proved in [5], we focus on finding an optimal solution in order to have a comparison point to latter design approximation algorithms. So, we propose an optimal static CRPD-aware preemptive scheduling algorithm in the sense that it minimizes the overall CRPD. To the best of our knowledge, no such optimal algorithm is known for this CRPD-aware scheduling problem or for scheduling with preemption delays.

The remainder of this paper is organized as follows: in Section II we define the CRPD-aware scheduling problem. Then, in Section III we detail our mathematical model and use it in Section IV to solve a simple example. Finally, we conclude our work in Section V and present further work.

II. Problem statement

In the remainder of this paper, we consider a finite set of n non recurring tasks τ_i, called jobs, that are defined by a release date r_i, a worst-case processing time p'_i, a deadline d_i and a worst-case preemption penalty s_i that is added to the worst-case processing time *when the job resumes after a preemption*. The objective is to compute a preemptive schedule that minimizes the overall preemption delays. Note that we assume in this paper that timing analyses are not subjected to any timing anomaly, which means that a cache hit always results in a shorter execution time than a cache miss.

Consider the following example with two periodic tasks T1 (respectively T2) with a WCET of 1 (resp. 7) and a deadline equals to its period of 3 (resp. 12). We assume a CRPD for T2 $s_2 = 0.5$. Note that such a value is not unrealistic: in real systems, the CRPD can be up to 20% of the task WCET as reported in [6]. Figure 1 presents an EDF schedule of 5 jobs generated by 2 tasks; with CRPD denoted s payed by the job τ_5 of task T_2. Note that for graphical representation ease, we depict the CRPD incurred by a preemption immediately after the task resumes its execution.

As soon as CRPD are considered, neither task-level fixed priority scheduling algorithms such as RM nor job-level ones such as EDF are optimal, as shown for example in [5]. Moreover, even the simplified problem of finding an optimal algorithm to schedule a finite set of jobs with a same CRPD for all jobs, has been proved in [5] to be NP-hard in the strong sense. Note that this problem is still NP-hard in the weak sense even if there are only two distinct release dates and deadlines and a preemption delay of one time unit as shown in [5].

However, in some corner cases, EDF does no generate any preemption, for example when all release dates (or deadlines) are equal or when release dates and deadlines are similarly ordered (i.e., $r_i \leq r_j \Rightarrow d_i \leq d_j$, $1 \leq i < j \leq n$). As a consequence EDF is an optimal online scheduling algorithm for the CRPD-aware scheduling problem under these specific conditions.

III. Mathematical model

Now, we define a mixed-integer linear program (MILP) to optimally solve the problem stated in Section II. Without loss of generality, we consider a slightly modified scheduling problem in which job processing times are defined as $p_i = p'_i - s_i$. In this latter scheduling problem, s_i can be interpreted as the delay incurred by a job *when it starts or resumes after a preemption*. Both problems are obviously equivalent.

The schedule will be considered as a set of slices delimited by subsequent job release dates or deadlines. Let $S = \{1, \dots m\}$ be the set of slices. The first slice begins by the smallest job release date whereas the slice m ends by the latest job deadline. Our approach is based on the following property:

Property 1: There exists an optimal schedule in which a job resumes at most once in every slice.

A direct consequence is that every job executed in a slice can pay at most one preemption delay. Thus, the previous property limits the number of schedule patterns to consider in order to define an optimal offline schedule in which all job deadlines are met.

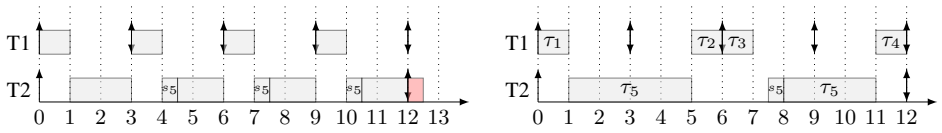
For every slice $k \in S$, b_k and e_k respectively denotes the slice starting time and ending time. The set of jobs that can be executed in a slice is denoted by $J_j, 1 \leq j \leq m$. For every job τ_i, we define the subsequent slices delimited by its release date and its deadline: $S_i \subseteq S$. Let $\bar{S}_i$ be the set S_i without its first slice. Thus, $S_i \setminus \bar{S}_i$ is the first slice of τ_i. Clearly, in any optimal schedule, the jobs are scheduled in a time interval delimited by its release date and its deadline. Jobs will be scheduled in slices and we call a job-piece the part of job executed in a given slice. τ_{ij} denotes the job-piece of τ_i executed in slice $j \in S_i$.

The main variables of our MILP formulation are:

- t_{ij} is a real variable the starting time of job-piece τ_{ij},
- p_{ij} is a real variable the processing time of job-piece τ_{ij},
- Δ_{ij} is a binary variable indicating if job-piece τ_{ij} has to pay a preemption delay s_i.

All notations are summarized in Table I.

The objective function is to minimize the over-



(a) EDF schedule (b) Optimal schedule

Figure 1: EDF and optimal schedules for Tasks T1$(1, 3, 3)$ and T2$(7, 12, 12)$

Notation	Type	Description
		Input data
S	Set	Set of slice indexes
S_i	Set	Slice indexes of τ_i
$\bar{S_i}$	Set	Slices of τ_i except the first one
$S_i \setminus \bar{S_i}$	Set	Index of the first slice of τ_i
J_j	Set	Jobs in slice j
p_i	real	Job τ_i processing time
s_i	real	Job τ_i preemption delay
b_j	real	starting time of slice j
e_j	real	ending time of slice j
		Output variables
t_{ij}	real	starting time of job-piece τ_{ij}
p_{ij}	real	processing time of job-piece τ_{ij}
Δ_{ij}	binary	preemption delay incurs by τ_{ij}
		Internal variables
a_{ij}	binary	condition $p_{ij} > 0$
b_{ij}	binary	condition $t_{ij} > t_{i,j-1} + p_{i,j-1} + s_i\Delta_{i,j-1}$
y_{ikj}	binary	job-piece disjunctive constraints

Table I: Data and variables for the MILP

all preemption delays among feasible schedule:

$$\min z = \sum_{i=1}^{n} \sum_{j \in S_i} s_i \Delta_{ij}$$

The MILP constraints are defined by different categories which are detailed hereafter (Equations 1 to 14).

A. Processing time constraints

The first set of constraints ensure all job-pieces of a job put together execute for exactly the execution time of that job.

$$\sum_{j \in S_i} p_{ij} = p_i \qquad 1 \leq i \leq n \qquad (1)$$

B. Slice constraints

Job-pieces τ_{ij} are executed inside a slice j. This means that every job-piece starts and completes in the interval $[b_j, e_j)$. We use an arbitrary small value ϵ to forbid a job-piece to start at time e_j (i.e., to avoid the case $t_{ij} = e_j$ and $p_{ij} = 0$).

$$t_{ij} + p_{ij} + s_i\Delta_{ij} \leq e_j \quad 1 \leq i \leq n, j \in S_i \quad (2)$$
$$t_{ij} \geq b_j \quad 1 \leq i \leq n, j \in S_i \quad (3)$$
$$t_{ij} \leq e_j - \epsilon \quad 1 \leq i \leq n, j \in S_i \quad (4)$$

Job-pieces executed inside every slice, including preemption delays, do not exceed the interval size.

$$\sum_{i \in J_j} p_{ij} + s_i\Delta_{ij} \leq e_j - b_j \qquad j \in S \qquad (5)$$

C. Job-piece disjunctive constraints

Inside every slice, two job-piece cannot be executed simultaneously. For every pair of job-pieces τ_{ij} and τ_{kj} we have either:

$$t_{ij} + p_{ij} + s_i\Delta_{ij} \quad \leq \quad t_{kj}$$

or

$$t_{kj} + p_{kj} + s_k\Delta_{kj} \quad \leq \quad t_{ij}$$

The previous disjunctive constraints can be linearized using the classical big M method and a binary variable y_{ikj}, $i < k$, set to 1 by the solver if τ_{ij} is executed before τ_{kj} in slice j, 0 otherwise:

$$t_{ij} + p_{ij} + s_i\Delta_{ij} \leq t_{kj} + (1 - y_{ikj})M \qquad j \in S_i \cap S_k \quad (6)$$
$$t_{kj} + p_{kj} + s_k\Delta_{kj} \leq t_{ij} + y_{ikj}M \qquad j \in S_i \cap S_k \quad (7)$$

D. Preemption penalty constraints

The binary variable $\Delta_{ij} = 1$ if job-piece τ_{ij} is subjected to a preemption delay in slice j, 0 otherwise. τ_{ij} is subjected to a preemption delay

in Slice j if $p_{i,j} > 0$ and (except for the first slice for Job τ_i) $\tau_{i,j-1}$ and τ_{ij} are not contiguous:

$$p_{ij} > 0 \qquad 1 \leq i \leq n, j \in S_i \setminus \bar{S}_i$$
$$p_{ij} > 0 \quad \text{and} \quad t_{ij} > t_{i,j-1} + p_{i,j-1} + \Delta_{i,j-1} \times s_i \qquad 1 \leq i \leq n, j \in \bar{S}_i$$

These conditions are linearized as follow by introducing binary variables a_{ij} and b_{ij}:

$$p_{ij} \leq \Delta_{ij} M \qquad 1 \leq i \leq n, j \in S_i \setminus \bar{S}_i. \quad (8)$$

and

$$p_{ij} \leq a_{ij} M \qquad 1 \leq i \leq n, j \in \bar{S}_i \quad (9)$$
$$t_{ij} - (t_{i,j-1} + p_{i,j-1} + \Delta_{i,j-1} \times s_i) \leq b_{ij} M \qquad 1 \leq i \leq n, j \in \bar{S}_i \quad (10)$$

According to these new binary variables, Δ_{ij} is defined by the logical result of $a_{ij} \wedge b_{ij}$ which can be linearised by computing $\Delta_{ij} = \min(a_{ij}, b_{ij})$:

$$\Delta_{ij} \leq a_{ij} \qquad 1 \leq i \leq n, j \in \bar{S}_i \quad (11)$$
$$\Delta_{ij} \leq b_{ij} \qquad 1 \leq i \leq n, j \in \bar{S}_i \quad (12)$$
$$\Delta_{ij} \geq a_{ij} + b_{ij} - 1 \qquad 1 \leq i \leq n, j \in \bar{S}_i \quad (13)$$
$$\Delta_{ij} \geq 0 \qquad 1 \leq i \leq n, j \in \bar{S}_i \quad (14)$$

IV. Application example

Consider again the example presented in figure 1, defined in Section II. As depicted in Figure 1a, T1 and T2 are not schedulable with EDF as T2 misses its deadline at Time 12, but the MILP will define a feasible one, depicted in Figure 1b.

For the MILP, we consider the different jobs $\tau_i(r_i, p'_i, s_i, d_i)$ from T1 and T2 over the hyperperiod which is equal to 12. On that interval of time, T1 issues 4 jobs $\tau_1(0, 1, 0.2, 2)$, $\tau_2(3, 1, 0.2, 2)$, $\tau_3(6, 1, 0.2, 2)$ and $\tau_4(9, 1, 0.2, 2)$, whereas T2 issues only one job $\tau_5(0, 7, 0.5, 12)$. Release dates and deadlines define four slices in the schedule: $[0, 3)$, $[3, 6)$, $[6, 9)$ and $[9, 12)$. τ_1 must be executed in the first slice, τ_2 in the second one, τ_3 in the third one and τ_4 in the last one, whereas τ_5 can be executed in any of them. For this example, we set $\epsilon = 0.01$ and $M = 100$.

The minimum of the objective function is $z = 3$ and the output variables computed by a solver (*CPLEX* 12.6.1 from IBM) are presented in Table II. Remember that the MILP solves a modified version of the CRPD-aware scheduling problem in which a preemption delay is paid the first time a job starts. Column p'_{ij} in Table II gives the processing time values depicted in the Gantt chart (Figure 1). In that figure, γ represents the preemption delay for Task T2 s_2.

In practice, the delay is spread over the task execution as an additional cost is incurred every time the task references a block that is no longer in the cache because of the interference due to the preempting task.

job	slice	t_{ij}	p_{ij}	Δ_{ij}	p'_{ij}
τ_1	1	0	0.8	1	1
τ_2	2	5	0.8	1	1
τ_3	3	6	0.8	1	1
τ_4	4	11	0.8	1	1
τ_5	1	2	1.5	1	2
τ_5	2	3	2	0	2
τ_5	2	7.5	1	1	1.5
τ_5	4	9	2	0	2

Table II: Output variables computed by the solver

.

V. Conclusion

Designing an optimal offline algorithm is a challenging problem. We have formulated the offline CRPD-aware scheduling problem as mixed-integer linear program. We intend to use it latter as a basis for performance evaluation and to evaluate the loss of schedulability of well-known on-line scheduling policy.

References

[1] S. Altmeyer, R. I. Davis, and C. Maiza, "Improved cache related pre-emption delay aware response time analysis for fixed priority pre-emptive systems," *Real-Time Systems*, pp. 499–526, 2012.

[2] F. Mueller, "Compiler support for software-based cache partitioning," in *ACM Sigplan Notices*, 1995, pp. 125–133.

[3] G. C. Buttazzo, M. Bertogna, and G. Yao, "Limited preemptive scheduling for real-time systems. a survey," *Industrial Informatics, IEEE Transactions on*, vol. 9, no. 1, pp. 3–15, 2013.

[4] J. Calandrino and J. Anderson, "On the design and implementation of a cache-aware multicore real-time scheduler," in *Real-Time Systems, 2009. ECRTS '09. 21st Euromicro Conference on*, 2009, pp. 194–204.

[5] G. Phavorin, P. Richard, and C. Maiza, "Complexity of scheduling real-time tasks subjected to cache-related preemption delays," LIAS, Université de Poitiers, Tech. Rep., 2015.

[6] W. Lunniss, S. Altmeyer, and R. Davis, "A comparison between fixed priority and edf scheduling accounting for cache related pre-emption delays," *Leibniz Transactions on Embedded Systems*, pp. 1–24, 2014.

Trading-off Data Consistency for Timeliness in Real-Time Database Systems

Simin Cai, Barbara Gallina, Dag Nyström, Cristina Seceleanu
Mälardalen Real-Time Research Centre, Mälardalen University, Västerås, Sweden
{simin.cai, barbara.gallina, dag.nystrom, cristina.seceleanu}@mdh.se

***Abstract*—In order to guarantee transaction timeliness, Real-time Database Management Systems (RTDBMSs) often relax data consistency by relaxing the ACID transaction properties. Such relaxation varies depending on the application and thus different transaction management mechanisms have to be decided for developing a tailored RTDBMS. However, current RTDBMSs development does not include systematic verification of timeliness and desired ACID properties. Consequently, the implemented transaction management mechanisms may breach timeliness of transactions. In this paper, we propose a process called DAGGERS for developing a tailored RTDBMS that guarantees timeliness and desired data consistency for real-time systems by employing model-checking techniques during the process. Based on the characteristics of the desired data manipulations, transaction models are designed and then formally verified iteratively together with selected run-time mechanisms, in order to achieve the desired/necessary trade-offs between timeliness and data consistency. The outcome of DAGGERS is thus a tailored transaction management with guaranteed appropriate trade-offs, as well as the model-checking based worst-case execution times and blocking times of transactions under these mechanisms and assumptions of the hardware architecture.**

***Keywords*—RTDBMS, timeliness, ACID, formal verification.**

I. Introduction

Real-time Database Management Systems (RTDBMSs) often serve as data management middleware for Real-Time Systems (RTSs), providing consistent data manipulation and guaranteed transaction timeliness. Traditionally, data consistency is the most important concern in non-real-time Database Management Systems (DBMSs), and ensured by transaction management. A transaction must guarantee the so-called ACID properties, that is, atomicity (a transaction either runs completely or rollbacks all changes), consistency (a transaction executing by itself must not violate logical constraints), isolation (uncommitted changes of one transaction should not be seen by concurrent transactions) and durability (committed changes are made permanent) [1]. In an RTDBMS, however, timely response of transactions must be guaranteed, even at the cost of data consistency [2]. In order to ensure timeliness, RTDBMSs often need to relax the ACID assurance [2].

The relaxation of ACID may differ from application to application, since many RTSs are implemented as special-purpose embedded solutions and thus entail high variability in both functional and extra-functional data management requirements [3]. Consequently, the designer of an RTDBMS must decide the specific run-time mechanisms for a particular real-time application, for instance the appropriate concurrency control mechanism and recovery mechanism, which ensure the desired/necessary relaxation of ACID.

However, during the development of RTDBMS, there exists no systematic method that guides the decisions of these run-time mechanisms so that the timeliness is guaranteed and the desired ACID relaxation is achieved. This may lead to an unpredictable system as well as unnecessary trade-offs. On the one hand, the decision of such mechanisms will inevitably impact the response time of transactions and may even result in unbounded execution or blocking time. Let us assume that, in order to ensure desired isolation, one needs to implement a concurrency control mechanism WAIT-50 [4] in the RTDBMS. This might introduce unbounded delay caused by transaction rollbacks and restarts due to conflicts. On the other hand, it is difficult to reason and verify if the relaxation of ACID is truly necessary. Consequently, we need an RTDBMS design process that could solve such challenges.

In this paper, we propose the DAGGERS process for developing a tailored RTDBMS that ensures transaction timeliness and desired data consistency. Our process allows system designers to systematically specify the trade-offs between global logical consistency and temporal constraints of transactions, from system requirements, and create a specialized transaction model with the desired properties. In order to ensure such properties, we carry out formal verification of transactional behaviors against the desired properties, which are formalized together with candidate run-time mechanisms for transaction management within the UPPAAL environment [5]. Finally, the selected run-time mechanisms that are proven to solve possibly existing conflicts are woven into the RTDBMS.

Our main contribution is two-fold. First, we provide the run-time mechanisms to tailor the RTDBMS such that it ensures the desired relaxation of ACID without breaching timeliness. Second, we provide the real-time properties such as worst-case execution time (WCET) and blocking time of the transactions, which account for the selected mechanisms in the final outcome under particularly stated hardware assumptions.

The paper is structured as follows: Section II describes the system model. The process is presented in Section III. The technical challenges, as well as the status of our research, are presented in Section IV. In Section V we present the related work. Finally we summarize the paper.

II. Assumed System Model

In this paper we consider a hard real-time system where all data access and manipulation go through transactions in an RTDBMS.

Logically related operations on data form work units. Each work unit WU_i includes a set of read and write operations in the database, as well as operations that perform calculation on data. Each work unit has a set of real-time requirements, including its invocation pattern and expected response time.

Each work unit WU_i is encapsulated as a transaction TR_i in the RTDBMS, associated with the desired ACID properties to be ensured. Each transaction also has a set of real-time properties, with a period (or minimal inter-arrival time) and

a relative deadline inherited from the work unit. The ACID assurance of transactions requires transaction management mechanisms to start, abort and rollback the transactions in a controlled manner, which in return affects the execution time and blocking time. The execution time of a transaction includes the execution time of operations in the work unit, plus the execution time of computation to ensure the ACID properties, such as lock acquisition and logging. A transaction may be blocked by concurrent transactions. The blocking time is determined by the concurrency control mechanism implemented in the RTDBMS.

At the scheduler level, transaction TR_i is executed as a task τ_i. A scheduling policy is adopted by the scheduler, which is not part of the RTDBMS. In order to achieve a predictable system, the execution time and blocking time have to be bounded, and the task set should be schedulable. These constraints, together with the desired ACID properties, form the constraints on the decision of run-time mechanisms implemented in the RTDBMS.

III. The DAGGERS Process

Our DAGGERS process is a design-time systematic process based on formal methods, meant to support system engineers in identifying, designing, and implementing adequate transactional support for RTDBMS in real-time systems. DAGGERS inherits the PRISMA [6] terminological framework as well as its methodological approach. In this section, we present the major steps of the DAGGERS process, as shown in Fig. 1. In the figure, A, C, I, D and T stand for atomicity, consistency, isolation, durability and timeliness, respectively. A specific relaxation of atomicity, also called a variant of atomicity [6], is denoted as A_i, while one specific instance of A_i is denoted as $A_{i,k}$. Similar denotations are applied to C, I, D and T variants. In this paper we focus on hard real-time systems, in which timeliness should never be relaxed (thus no other variants of T). But in principle the process can be applied to mixed criticality RTSs, where multiple variants of timeliness may exist.

We start with specifying the operations in the work units and their logical and temporal properties. Then we formalize them into transaction models, and verify the timeliness and desired ACID properties with selected run-time mechanisms. Finally, under particular hardware behavior assumptions/models, we obtain a set of tasks with their WCETs and blocking times, which can be used for schedulability analysis, as well as the verified run-time mechanisms for a tailored RTDBMS.

In the following we describe the steps of the process.

Step I: From system requirements a data management expert identifies the work units, which are partially ordered sets of logically related operations on data items, as well as the logical and temporal requirements characterizing these work units. The initial transactional properties are then specified based on these requirements. A set of desired properties is associated to each work unit, respectively, consisting of certain variants of ACID and the timeliness constraints. For better illustration, we assume that the desired isolation variant I_i is identified to be PL-3 isolation level, which ensures full isolation [7]. These specified computational behaviors together with the properties are called *transaction types*.

Step II: The second step of DAGGERS consists of the building, iterative tailoring and refinement of the transaction models. First, by analyzing the dependencies between transaction types, the data management expert organizes the transaction types according to well-known transaction model types. Next, initial yet formal design specifications are created, including the design-level decisions for the ACID variants, such as an optimistic concurrency control algorithm for PL-3 isolation, denoted as $I_{i,k}$. The specifications can then be modeled and verified by model-checking tools.

We propose to model transactional behavior as networks of timed automata [8], which have underlying formal semantics implemented in the state-of-the-art model-checker for real-time systems, called UPPAAL [5]. In order to find possible conflicting behaviors, we model-check the transaction models annotated with timing information in form of invariants. The output of the verification is given in terms of "yes/no" answers, but also in terms of timing properties of transactions (e.g., execution time, blocking time, etc.). In case of a "no" answer (meaning that inconsistencies exist) the verification returns a counter-example that exposes the respective inconsistency. The verification and validation of the transactions can be used to detect: (i) any behavioral inconsistencies between the stated transactional properties and the logical data requirements, as well as (ii) any unresolvable timing inconsistencies between the potential conflicts among concurrently executed transactions and the timing requirements.

The transaction models are checked iteratively, together with different candidate run-time mechanisms, provided by the platform. If model-checking shows that unsolvable conflicts occur with a particular candidate run-time mechanism (e.g., optimistic concurrency control), the mechanism will be replaced by another candidate (e.g., a lock-based algorithm), and the model is verified again. Ideally, the successive checking stops when no conflicts are detected anymore.

In case inconsistencies that cannot be resolved by any candidate run-time mechanism model exist, traces of the inconsistency as well as other feedback from the verification will be stored into a repository. The obtained feedback can act as heuristics that may help the data management expert to change the transaction models, adjust the scheduling policy to separate such transactions, or reflect further on the requirements. For instance, a less restrictive isolation variant such as PL-1 isolation level [7] might be considered to replace the PL-3 isolation, since deadlines will be missed if the system has to ensure the current logical consistency requirements. Then the modeling and model-checking will restart. This iteration continues until a consistent formalization is achieved.

At the end of this step, a set of refined transaction models is generated, with specified timing constraints such as transaction deadlines, and appropriate variants of atomicity, consistency, isolation and durability. The WCET and blocking time for each transaction are obtained with these selected mechanisms, under particular hardware assumptions (e.g., we can assume a hardware whose impact on the WCET is so small that it can be ignored, or we can use abstract hardware architecture (including cache) models in UPPAAL). In addition to transaction models, the output of this step also includes the potential conflicts, and the corresponding run-time mechanisms able to resolve these conflicts, for each transaction model respectively.

Step III: In this step, we form a task set with their WCETs and blocking times. In addition, we automatically generate an RTDBMS by composing the selected run-time mechanisms. We propose to generate the RTDBMS using the COMET (COMponent-based Embedded real-Time) [9] platform, which consists of a set of components that encapsulate specific real-time database functionalities, and a set of aspects that encapsulate cross-cutting concerns such as concurrency control

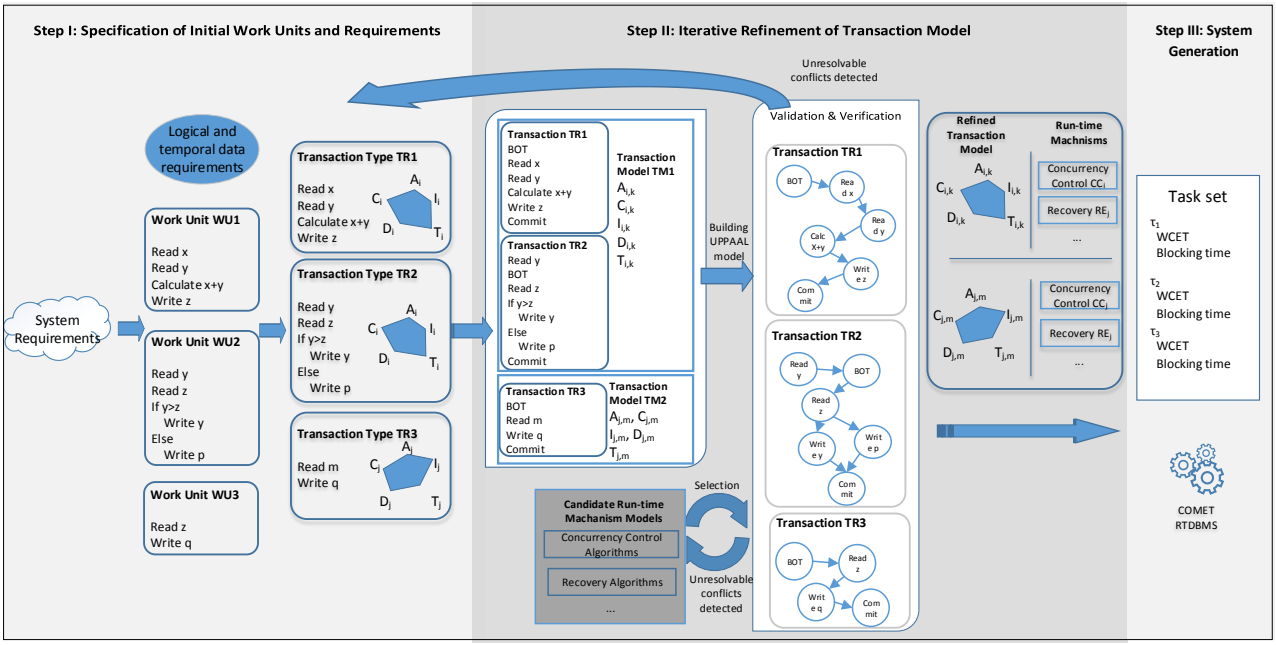


Fig. 1. The DAGGERS process

and logging. By weaving selected components and aspects from COMET, we generate a tailored RTDBMS that provides run-time support for the resolution of potential conflicts and assurance of desired transactional properties.

IV. Technical Challenges and Status of Research

Several key challenges need to be addressed in order to support the DAGGERS process. One major challenge is specifying and modeling the real-time transaction models. The ACID and timeliness properties should be specified, in such a way that we can easily reason about the possible relaxation of ACID for timeliness. Existing techniques either only focus on ACID, or have limitation in formal syntax and semantics. The properties need to be specified in Timed Computation Tree Logic (TCTL), and the behaviors of transactions as networks of timed automata in UPPAAL. During the iterative verification step, different candidate run-time mechanism models need to be woven into the timed automata network dynamically, which is not a trivial task. Guidelines and tool support, for example specification patterns [10], are desired for the correctness and efficiency of modeling.

Obtaining the WCETs and blocking times of transactions is another challenge. Although some work has been done, for example by Gustavsson et. al [11], on WCET analysis for concurrent programs using UPPAAL, it remains nontrivial to model and analyze database transactions with ACID properties.

Another challenge is to find a predictable resolution mechanism for the detected conflicts. In some situations, conflicts may be resolvable without missing any deadlines if transactions with certain dependencies are grouped together and different run-time mechanisms are applied on different groups. Methods for grouping transactions are necessary, which are performed by either the verification tool, or the database expert. It is also possible that the conflicts among some transactions are not resolvable within bounded time by any run-time mechanisms. For instance, no concurrency control mechanism can ensure the desired isolation of particular transactions without breaching their deadlines. In this case, these transactions have to be separated in time, using techniques such as static scheduling [12], offset scheduling [13] or explicit mutual exclusion by semaphores in the transactions. In order to adjust the scheduling policy, the verification framework should provide traces of the conflicts, as well as other information such as the blocking time from conflicting transactions and rolling-back/restarting time caused by the conflicts.

One common challenge for solutions using model-checking techniques is the scalability of model checking. UPPAAL-SMC, an extension of UPPAAL with statistical model checking for priced timed automata [14], improves the scalability via bounded model-checking and can be used for the bounded verification of complex systems, assuming certain probability distributions. However, the provided result is not a guarantee as in the symbolic model-checking, but rather a probability estimation of a certain property, with a specific accuracy.

Regarding the status of our research, we have already carried out some preliminary work on modeling transactions with particular concurrency control mechanisms in UPPAAL [15], and will continue to involve other mechanisms and properties. Currently we are also working on a taxonomy of real-time data aggregation, with a focus on the trade-offs between transaction properties during the aggregation process. In our next steps, we plan to develop a formal language for specifying real-time transaction properties from system requirements. We will then develop a framework based on UPPAAL, including models of common candidate run-time mechanisms, for building and iterative verification of transaction models. Finally, we plan to implement an RTDBMS with the tailored transaction management using the COMET platform.

V. Related Work

Relaxing ACID for a compromised consistency has been investigated in the database community. In recent work, nec-

essary trade-offs have been proposed among data consistency, availability and scalability [16], as well as database performance [17]. However, existing works do not take timeliness into consideration, which is a core property for RTSs and the focus of our process.

Noticeable research efforts have been made in engineering tailored DBMSs in recent years. COMET [9] combines a component-based approach and aspect-oriented programing to build tailored RTDBMS. Encapsulating database functionalities as components, and crosscutting features as aspects, COMET generates a tailored RTDBMS by weaving the selected components and aspects together. In FAME-DBMS [3], functional requirements on a DBMS are represented as features. DBMS variants are generated by composing the reusable features. The selection of building modules in these approaches is based on functional requirements analysis, as well as constraints on code-size and performance. They mainly address resource consumption and footprint issues for embedded systems, rather than the timeliness of transactions and the possible conflicts with ACID assurance. Our process, as a contrast, starts from deriving the real-time transaction models accounting for both timeliness and ACID properties. During the verification of transaction models, the run-time implementations are selected, and are proved to guarantee the desired timeliness and data consistency requirements. The existing platforms for generating the tailored DBMS can be integrated into our process at the implementation phase.

Substantial work has been done on formal specification and reasoning of transaction models. The ACTA [18] framework provides a first order logic formalization to specify the transactional effects on data and the interaction between transactions, facilitating reasoning of transaction properties and flexible synthesis of transaction models. Real-Time ACTA [19] extends ACTA with formalization of real-time constraints on transactions and data. However, the formal syntax and semantics for specification of ACID variants provided by ACTA and Real-Time ACTA are limited, and tool-support for verification is lacking. SPLACID [20] improves ACTA by providing a more complete language support for ACID variants and their subfeatures, but real-time properties are not included. Non-ACTA descendant work has also been proposed. For example, Wang et al. [21] proposed Abstract Transaction Construct (ATC) that encapsulates the structure and behavior of a transaction service. However, none of these works provide specification for both ACID and real-time properties, or support verification of transaction models with run-time mechanisms.

VI. Summary

In this paper, we have introduced the DAGGERS process (of our newly funded project DAGGERS), which aims at generating a tailored RTDBMS that ensures appropriate tradeoffs between global data consistency and transaction timeliness in RTSs. In this process, work units and their ACID and timeliness properties are specified from system requirements. Based on this specification, transaction models are derived and iteratively refined by means of formal verification. By composing the run-time mechanisms proved to resolve possible detected conflicts, an RTDBMS is generated to support the refined transaction models. A task set with their WCETs and blocking times are obtained.

Compared to current tailored DBMS solutions that commonly base their customization on the functional requirements, and constraints on resource consumption, footprint and performance, our process relies on the analysis and verification of transaction models, thus achieving the desired data consistency and transaction timeliness required by the particular RTS.

References

[1] J. Gray and A. Reuter, *Transaction Processing: Concepts and Techniques*, 1st ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1992.

[2] J. A. Stankovic, S. H. Son, and J. Hansson, "Misconceptions about real-time databases," *Computer*, vol. 32, no. 6, pp. 29–36, 1999.

[3] M. Rosenmüller, N. Siegmund, H. Schirmeier, J. Sincero, S. Apel, T. Leich, O. Spinczyk, and G. Saake, "Fame-dbms: tailor-made data management solutions for embedded systems," in *Proceedings of the 2008 EDBT workshop on Software engineering for tailor-made data management*. ACM, 2008, pp. 1–6.

[4] J. Haritsa, M. Carey, and M. Livny, "Dynamic real-time optimistic concurrency control," in *Real-Time Systems Symposium, 1990. Proceedings., 11th*, Dec 1990, pp. 94–103.

[5] K. G. Larsen, P. Pettersson, and Y. Wang, "Uppaal in a nutshell," *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 1, no. 1, pp. 134–152, 1997.

[6] B. Gallina, "Prisma: a software product line-oriented process for the requirements engineering of flexible transaction models," Ph.D. dissertation, University of Luxembourg, 2010.

[7] A. Adya, B. Liskov, and P. O'Neil, "Generalized isolation level definitions," in *Data Engineering, 2000. Proceedings. 16th International Conference on*, 2000, pp. 67–78.

[8] R. Alur and D. L. Dill, "A theory of timed automata," *Theoretical computer science*, vol. 126, no. 2, pp. 183–235, 1994.

[9] D. Nyström, A. Tešanovic, M. Nolin, C. Norström, and J. Hansson, "Comet: a component-based real-time database for automotive systems," in *Proceedings of the Workshop on Software Engineering for Automotive Systems*. IET, 2004, pp. 1–8.

[10] S. Konrad and B. H. C. Cheng, "Real-time specification patterns," in *Proceedings of the 27th International Conference on Software Engineering*. ACM, 2005, pp. 372–381.

[11] A. Gustavsson, A. Ermedahl, B. Lisper, and P. Pettersson, "Towards wcet analysis of multicore architectures using uppaal," in *10th International Workshop on Worst-Case Execution Time Analysis*, 2010, pp. 101–112.

[12] G. C. Buttazzo, *Hard real-time computing systems: predictable scheduling algorithms and applications*. Springer Science & Business Media, 2011, vol. 24.

[13] J. Palencia and M. Gonzalez Harbour, "Schedulability analysis for tasks with static and dynamic offsets," in *Real-Time Systems Symposium, 1998. Proceedings., The 19th IEEE*, Dec 1998, pp. 26–37.

[14] A. David, K. Larsen, A. Legay, M. Mikučionis, D. Poulsen, J. van Vliet, and Z. Wang, "Statistical model checking for networks of priced timed automata," in *Formal Modeling and Analysis of Timed Systems*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2011, vol. 6919, pp. 80–96.

[15] S. Cai, "Modeling real-time transactions in uppaal," Tech. Rep., April 2015. [Online]. Available: http://www.es.mdh.se/publications/3911-

[16] D. Pritchett, "Base: An acid alternative," *Queue*, vol. 6, no. 3, pp. 48–55, May 2008.

[17] D. J. Abadi, "Consistency tradeoffs in modern distributed database system design: Cap is only part of the story," *Computer*, vol. 45, no. 2, pp. 37–42, 2012.

[18] P. Chrysanthis and K. Ramamritham, "Synthesis of extended transaction models using acta," *ACM Transactions on Database Systems (TODS)*, vol. 19, no. 3, pp. 450–491, 1994.

[19] M. Xiong and K. Ramamritham, "Specification and analysis of transactions in real-time active databases," in *Real-Time Database and Information Systems: Research Advances*. Springer US, 1997, vol. 420, pp. 327–351.

[20] B. Gallina and N. Guelfi, "Splacid: An spl-oriented, acta-based, language for reusing (varying) acid properties," in *Software Engineering Workshop (SEW), 32nd Annual IEEE*, 2008, pp. 115–124.

[21] T. Wang, P. Grefen, and J. Vonk, "Abstract transaction construct: Building a transaction framework for contract-driven, service-oriented business processes," in *Service-Oriented Computing–ICSOC 2006*. Springer, 2006, pp. 434–439.

Performance evaluation of a distributed IMA architecture

Emilie Deroche*†, Jean-Luc Scharbarg† and Christian Fraboul†
*Airbus Group SAS, Airbus Group Innovations, 18, rue Marius Terce, 31025 Toulouse, France
Email: {Emilie.Deroche}@airbus.com
†University of Toulouse, IRIT-INPT/ENSEEIHT, 2, rue Charles Camichel, 31000 Toulouse, France
Email: {Emilie.Deroche, Christian.Fraboul, Jean-Luc.Scharbarg}@enseeiht.fr

***Abstract*—Current avionics architectures implemented on large aircraft use complex processors, which are shared by many avionics applications according Integrated Modular Avionics (IMA) concepts. Using less complex processors on smaller aircraft such as helicopters leads to a distributed IMA architecture. The objective of this paper is to evaluate, on a realistic helicopter case study, the feasibility of such distributed architectures. Main contribution is the evaluation of the allocation of the same reference avionics application on different possible distributed physical architectures. Managing the allocation of the avionics application, described as ARINC 653 partitions linked by APEX logical communication channels is twofold. A first problem deals with the feasibility of a static allocation of partitions on each processing unit and the static allocation of logical communication channels on communication links or shared networks. But a second problem is the worst-case end-to-end distributed application delay analysis: due to the scheduling of partitions on each asynchronous processor and the scheduling of communication on shared communication means, some allocation schemes cannot be chosen. This paper points out the complexity of taking into account these two aspects altogether on different possible distributed target architectures.**

***Keywords*-Distributed Integrated Modular Avionics; distributed processors allocation problem; end-to-end applicative delay analysis;**

I. Introduction

Helicopter and aircraft industries attempt to reduce weight and power consumption in helicopters and aircraft. The Integrated Modular Avionics (IMA) architecture is a first step in this direction: instead of having one function per processor like in federated architectures, several functions share the same processor. Moreover, communication can be also shared to reduce the number and the weight of cables [8][12]. But the implementation of IMA in helicopters has to take into account additional constraints.

In large aircraft, processing units are grouped in a limited number of centralized racks [12]. But in smaller aircraft such as helicopters, the idea is to integrate equipment in unused area. Moreover new devices have to be positioned so as to balance weight in the whole helicopter [1].

To face these new constraints, one way is to have a larger number of (possibly less complex) processors that can be distributed in the whole helicopter. The problem is then to guarantee timing properties of distributed avionics systems.

Main objective of this paper is to evaluate the feasibility of such distributed IMA architecture on a realistic helicopter case study. Main contribution is the comparison of possible allocation schemes of the same reference avionics application on different physical architectures. Static allocation of the avionics application on a given architecture is based on necessary (but not sufficient) conditions for sharing processors and communication links. But dynamic aspects that lead to variable applicative end-to-end delay need a worst-case analysis.

II. Avionics system architectures context

Each avionics embedded system is composed of a set of functions distributed on a set of devices. Thus, an avionics system can be described according two aspects: an applicative architecture which describes how different functions can be grouped in communicating partitions and a physical architecture which describes how the different processing units can be interconnected through communication links or shared networks.

A. Physical architecture

Figure 1 depicts the physical architecture of a Vehicle Monitoring System (VMS) [2] we will use as a case study. Its goal is to alert the pilot when a parameter is starting to exceed limits before reaching the alarm threshold and to provide the value of a parameter when the pilot asks for it [3]. This typical avionics system is composed of two duplex Aircraft Management Computers (AMCs) [2], four Multi-Function Displays (MFDs) and local I/Os [2].

All these devices are interconnected thanks to dedicated point-to-point links (figure 1 example) or a shared network.

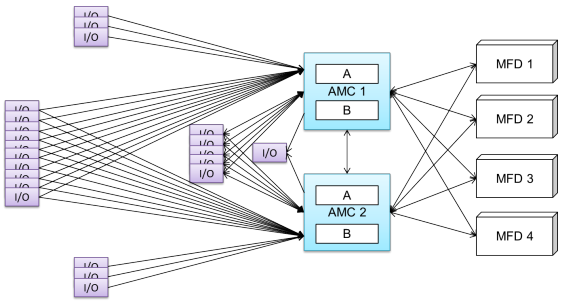


Figure 1. Reference physical architecture of the VMS

B. Applicative architecture

Each avionics function is modeled by a set of communicating partitions standardized by ARINC 653 [4]. A partition can be seen as the allocation unit on one processor. Four real-time attributes are associated to each partition P_i: period (duration between two activations, Per(i)), Worst Case Execution Time (WCET(i)), release time of the first occurrence of the partition (Offset, Rel(i)) and deadline as illustrated in figure 2.

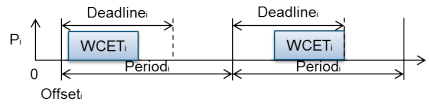


Figure 2. Real-time attributes of a partition P_i

Two partitions can communicate through APplication-EXecution (APEX) communication channels as described by ARINC 653 [4]. Figure 3 describes seven partitions that have to be allocated on each AMC processor of the Vehicle Monitoring System (VMS) physical architecture presented in figure 1.

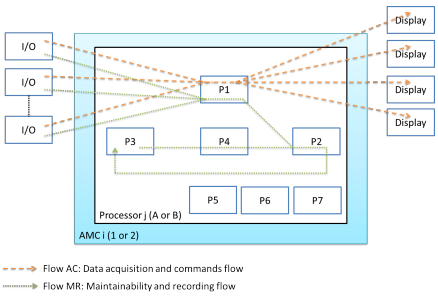


Figure 3. Main communication flows of communicating partitions

III. Allocation problem

A. Necessary conditions on processors allocation

Let us consider the Vehicle Monitoring System previously described. Its applicative architecture is depicted in figure 3. It includes 7 partitions, P_1 to P_7. Partitions P_1 to P_4 execute on 4 processors for redundancy reasons. Partitions P_5 to P_7 are used for recording and they execute on all the processors where at least one partition executes. This existing allocation is depicted in figure 4 (1 processor, allocation 1). 1 single processor out of the 4 redundant ones is represented. The 3 other ones are identical. One envisioned evolution of this system is to replace each processor by a set of less powerful processors. It will lead to 4 sets of processors. The set of partitions is identically distributed on each set of processors.

Figure 4 depicts 4 among 14 possible allocation schemes on sets of 2, 3 and 4 processors. Once again, only one set is represented for each possible allocation, since the 3 other sets are identical.

Let us have a look at allocation 2A. It considers 2 processors for each set. Partition P_1 is allocated to the

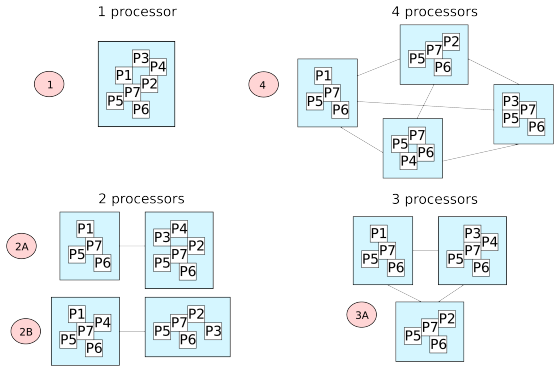


Figure 4. Possible allocation schemes

first processor while P_2, P_3 and P_4 are allocated to the second one. P_5, P_6 and P_7 have to be allocated to both processors of the set, as previously explained.

Once the different allocations are obtained, the MAjor Frames (MAFs) of each processor has to be built. These constructions consist in defining the possible hyper-cycles of the execution of partitions. As partitions are periodic, a pattern appears as illustrated in figure 5. This pattern lasts an hyper-period Hyp, i.e. the least common multiple (LCM) of the periods of the partitions allocated on the processor. It is split in different slots which each lasts the smallest period of the partitions allocated in the processor. Figure 5 shows one possible minimal MAF in one processor. A MAF is not retained if a partition cannot be scheduled.

This construction has to take into account the use of less powerful processors. Let us consider a processor which is less powerful than the initial processor by a factor α. We denote $WCET_\alpha(i)$ the WCET of partition P_i on this new processor. We assume that $WCET_\alpha(i)$ only depends on $WCET_{initial}(i)$ and α. Thus, we have:

$$WCET_\alpha(i) = \frac{WCET_{initial}(i)}{\alpha}, \alpha \in]0,1] \qquad (1)$$

This is an abstraction done in this work for reaching a panel of platforms. For a specific platform, WCETs will be determined for each partition on each processor.

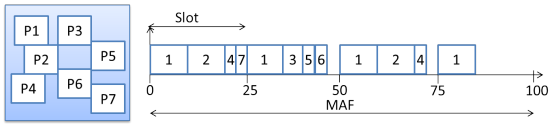


Figure 5. Sharing of processing ressources

B. Necessary conditions on communications allocation

The scheduling of messages highly depends on the kind of communication means. In the context of this case study, dedicated links between processors are considered. Each link is shared by the set of partitions allocated to the source processor of the link, based on a First Come First Served policy.

The transmission time $T_{m,l}$ of a given message m on a link l depends on both the size of the message $size(m)$ and the bandwidth of the link β. Classically, we have:

$$T_{m,l} = \frac{size(m)}{\beta(l)}, \beta(l) \in]0,1] \quad (2)$$

Let us denote $\mathcal{M}_l$ the set of all messages transmitted on link l. We have to ensure that l is not overloaded. In order to do that, we compute the overall bandwidth of all the messages in set $\mathcal{M}_l$. The bandwidth $Bd(m,l)$ for one message m depends on its transmission time $T_{m,l}$, the period $Per(i)$ of its source partition P_i and the bandwidth of the link $\beta(l)$.

$$Bd(m,l) = \frac{T_{m,l} * \beta(l)}{Per(i)} \quad (3)$$

Then, the overall bandwidth $Bd(l)$ for messages in $\mathcal{M}_l$ is:

$$Bd(l) = \sum_{m \in \mathcal{M}_l} Bd(m,l) \quad (4)$$

A given link l is not overloaded if $Bd(l)$ does not exceed the available bandwidth, i.e:

$$Bd(l) \leq \beta(l) \quad (5)$$

In this paper, we have presented architectures where the connection of processors is done by dedicated communication links. This connection can also be done by shared communication network. The difference impacts the worst case transmission time between a source partition P_i and a destination partition P_j ($WCTT(i,j)$): with a switched network, a jitter (present with dedicated links) can be more important due to a waiting time in switches, with a bus network, a bus access delay can exist, etc. Our final objective is to compare new distributed architectures taking into account the interconnection mean.

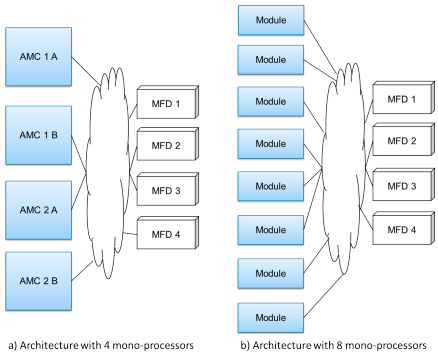


Figure 6. Example of new possible architectures

Figure 6 (part a) illustrates a possible evolution of the existing two bi-processors architecture: four mono-processors interconnected through a shared communication network. Figure 6 (part b) illustrates a more distributed architecture: eight mono-processors, interconnected by a shared network as illustrated in figure 6. The goal is to allocate partitions in different architectures as in figure 6 and check that necessary conditions as well as sufficient condition on application end-to-end delay presented in the next section are verified. The final objective is to compare new distributed architectures.

C. Sufficient condition on application end-to-end execution delay

Figure 7 presents the configurations (according α_{min} and β_{min} values) verifying necessary conditions presented in previous paragraphs.

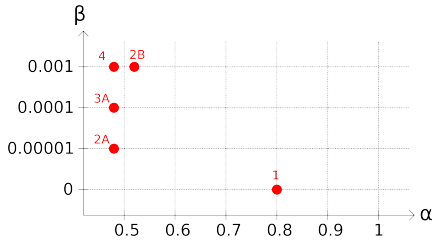


Figure 7. Minimal configurations allocation

However an allocation is valid only if the application end-to-end execution delay does not exceed a maximum value.

The application end-to-end execution delay corresponds to the maximum data age of an emitted data not overwritten, i.e. the maximum delay between the last produced input not overwritten until its last use. This delay is illustrated in figure 8 between P_2, P_3 and P_4 in the case of flow MR depicted in figure 3.

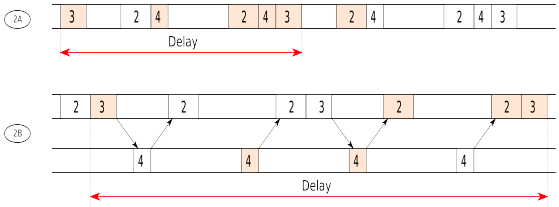


Figure 8. End-to-end execution delay with $\alpha = 0.6$ and $\beta(l) = 0.001$

According to the allocation and the scheduling of partitions, the end-to-end execution delay varies as depicted in figure 8. The worst case end-to-end execution delay depends on MAFs and offsets, i.e. the release time of processors: two previous consecutive partition instances cannot become reachable anymore as illustrated in figure 9.

These worst case offsets are obtained when a data arrives just after the beginning of the execution of the partition, implying the processing of data one period after.

In our study, sufficient condition is that the end-to-end execution delay cannot exceed one WCET and one period of the partition P_3. The evaluation of the worst end-to-end delay in figure 8 shows that the allocation 2A respects all the conditions whereas the allocation 2B is not valid. This preliminary study shows the importance of

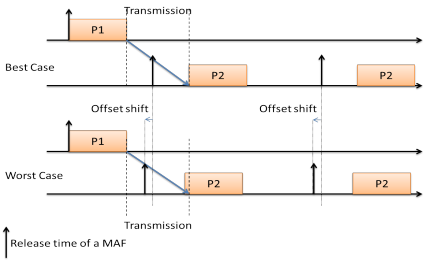


Figure 9. Influence of offsets on end-to-end delay

such a sufficient condition for validating a given allocation scheme.

D. The need of an integrated approach

The allocation process described in the previous section has to cope with two aspects: the first one is static allocation of partitions to processors and mapping of communication channels on available links and networks, the second one deals with the temporal analysis of applicative end-to-end delay according to a given allocation scheme. Previous works have been devoted to some of these problems.

Many of them take into account allocation problems. For certification and reliability reasons, the scheduling is done off-line. Thus, a planning is done on the basis of the temporal characteristics of partitions. In [5] the authors proceed in two steps : first they schedule partitions on execution nodes, second they route flows on the avionics network. Work proposed in [9] deals with the allocation of partitions trying to minimize the communication costs.

Some of them integrate temporal analysis of embedded systems. Many works have been devoted to the worst-case delay analysis on AFDX network [7][11], the temporal requirements verification of systems [10], and the complexity of communication delays [6].

Our aim is to propose an approach, which copes altogether with static allocation of partitions and logical communication channels and with a guaranteed dynamic applicative end-to-end delay. In the preceding sections we formalize the different parts of this approach.

IV. Conclusion

This paper shows how it could be possible to distribute an IMA architecture in order to cope with constraints of small aircraft or helicopters. Starting from an applicative architecture including a set of communicating partitions, the goal is to find the best physical architecture as well as the allocation which optimize given criteria. These criteria concern mainly processor speed and communication mean bandwidth. But the validation of temporal property is essential.

We are formalizing the allocation problem and work underway deals with the generalization of sufficient condition in order to validate more complex distributed avionics architecture (shared networks, remote I/Os). Moreover, main problem is to test all potential candidate architectures. This can be done only for small systems. Thus, a heuristic approach has to be defined in order to be able to find optimal or near-optimal allocation in the context of complex systems.

References

[1] *Aircraft Weight and Balance Handbook (FAA-H-8083-1A)*. AFS-630, P.O. Box 25082, Oklahoma City, OK 73125: The U.S. Department of Transporation, Federal Aviation Administration, Airmen Testing Standards Branch, 2007.

[2] "Aircraft accident report 1/2011 - report on the accident to eurocopter ec225 lp super puma, g-redu near the eastern trough area project (etap) central production facility platform in the north sea on 18 february 2009," Department for Transport, Tech. Rep., September 2011.

[3] "Helicopter companies look into 2014's crystal ball," www.helicomag.com/2014/01/04/helicopter-companies-look-into-2014's-crystal-ball/, January 2014, accessed: 09 February 2015.

[4] *Avionics application software interface, part 1 - Required services, ARINC specification 653P1-3*, Aeronautical radio, Inc., Aeronautical radio, Inc. Std., November 2010.

[5] A. Al Sheikh, "Resource allocation in hard real-time avionic systems. scheduling and routing problem," Ph.D. dissertation, EDSYS, September 2011.

[6] N. Badache, K. Jaffres-Runser, J.-L. Scharbarg, and C. Fraboul, "End-to-end delay analysis in an integrated modular avionics architecture," in *Emerging Technologies Factory Automation (ETFA), 2013 IEEE 18th Conference on*, Sept 2013, pp. 1–4.

[7] H. Bauer, J. Scharbarg, and C. Fraboul, "Worst-case end-to-end delay analysis of an avionics afdx network," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, March 2010, pp. 1220–1224.

[8] P. Bieber, F. Boniol, M. Boyer, E. Noulard, and C. Pagetti, "New challenges for future avionic architectures," *Journal Aerospace Lab*, vol. 4, May 2012.

[9] C. Ekelin and J. Jonsson, "A lower-bound algorithm for minimizing network communication in real-time systems," in *Parallel Processing, 2002. Proceedings. International Conference on*, 2002, pp. 343–351.

[10] M. Lauer, J. Ermont, F. Boniol, and C. Pagetti, "Latency and freshness analysis on ima systems," in *Emerging Technologies Factory Automation (ETFA), 2011 IEEE 16th Conference on*, Sept 2011, pp. 1–8.

[11] X. Li, J. Scharbarg, and C. Fraboul, "Worst-case delay analysis on a real-time heterogeneous network," in *Industrial Embedded Systems (SIES), 2012 7th IEEE International Symposium on*, June 2012, pp. 11–20.

[12] J. Moore, *Digital Avionics Handbook, Second Edition - 2 Volume Set*. CRC Press 2000, 2001, ch. Advanced distributed architectures, pp. 33–1–33–12.

Retargetable Infeasible Path Detection for WCET Analysis

Zachary S. Hall, Kory E. Kraft, Thomas D. Mincher, Joseph H. Iannetta, Christopher A. Healy
Department of Computer Science
Furman University
Greenville, SC, USA

***Abstract*—In worst-case execution time (WCET) analysis, often the input program contains loops or functions that have multiple execution paths. The paths have varying execution times, and it is the job of a WCET tool to identify worst-case paths. However, in some cases the worst-case path is infeasible. A timing tool can provide tighter bounds on the WCET if it can disregard infeasible paths from its analysis. This paper describes a new technique where infeasible paths are detected by means of branch analysis on the input program's assembly code. By doing the analysis at the assembly level makes it unnecessary to rely on a compiler to provide control-flow information. As a result, retargeting the timing tool to a new instruction set can be streamlined.**

Keywords—control-flow analysis, branch constraints, infeasbile paths, worst-case execution time

I. Introduction

The goal of worst-case execution time (WCET) analysis is to provide a real-time system with an accurate and tight bound on the WCET of tasks, so that the system can most effectively schedule these tasks. Quite often, an input program or benchmark being analyzed will feature one or more loops or functions that contain multiple execution paths. These paths come about due to the presence of conditional execution (e.g. if) statements, which give rise to conditional branch instructions (e.g. beq, blt, etc.) in the assembly code.

A WCET analysis tool needs to identify the worst-case path through each loop and function. However, it is possible that the longest path cannot actually be taken because it is infeasible. This can happen if the path traverses two branch instructions that require contradictory conditions to be true. For example, a function could contain two if-statements, one testing if a variable is zero, and the other if the same variable is not zero. Assuming that the variable is not updated, it would be impossible to enter the bodies of the two if-statements. Thus, we have an infeasible path. If a WCET tool can recognize infeasible paths and remove them from the analysis, it can realize two benefits. First, the complexity of the analysis is reduced, since fewer paths need to be considered. Second and more importantly, the WCET can be tightened whenever the longest-running path can be eliminated.

This paper describes a tool called RALPHO [5]: a Retargetable Assembly-Level Program Hierarchical Organizer. The purpose of RALPHO is to analyze assembly code to glean control-flow information. The authors have modified RALPHO to include branch constraint analysis in order to make it possible for our timing analysis tool [6, 7] to detect infeasible paths. It can work with any type of WCET analysis, whether it be traditional [10], parametric [4], or stochastic [8].

The contribution of this paper is that RALPHO is able to perform its analysis on assembly code, and it is relatively straightforward to retarget it to a new instruction set. In the past, our timing analysis tool relied on a compiler [2, 7] to emit control-flow information. To retarget the tool to a new architecture would require a significant modification to the compiler, which is a significant undertaking. RALPHO is also computationally efficient. It can determine the number of iterations of a loop in most cases, and pass this information to the timing analyzer, which can replicate the effect of each loop iteration [4, 6] in constant time. Other approaches that detect infeasible paths [3, 9, 10] perform a less computationally efficient analysis because the effect of each loop iteration needs to be explicitly evaluated one at a time.

II. Approach

Figure 1 depicts the framework for obtaining the WCET for a task, and RALPHO's role in the process. A compiler, such as gcc, generates assembly code for the target architecture. This assembly file is fed to the RALPHO system in order to determine the control-flow information. RALPHO also reads an Assembly Definition File (ADF), which specifies the format of the assembly syntax. Once the control-flow information has been gathered, it is then possible to enumerate the function instances and the set of paths for each loop and function in the program. In this paper, we focus on the control-flow information phase of RALPHO.

RALPHO presents its control-flow information hierarchically. A program consists of functions, which in turn consists of instructions. The instructions can be organized into basic blocks. Some basic blocks contain a transfer of control instruction, which gives rise to control flow and loops. The current design of RALPHO is designed for the ARM7 platform but it also retargetable other instruction sets. Details about RALPHO's original design can be found in [5].

The structure of our timing analysis tool is beyond the scope of this paper. The timing analyzer expects input files, including information about the program's control flow from RALPHO, to conform to a predefined format. It is then

This work was supported in part by grants from the Furman Advantage and Francis M. Hipp Research Fellowship and the South Carolina Independent Colleges and Universities Research Program.

capable of estimating the WCET, the parametric WCET when appropriate, and the probabilistic distribution of execution times.

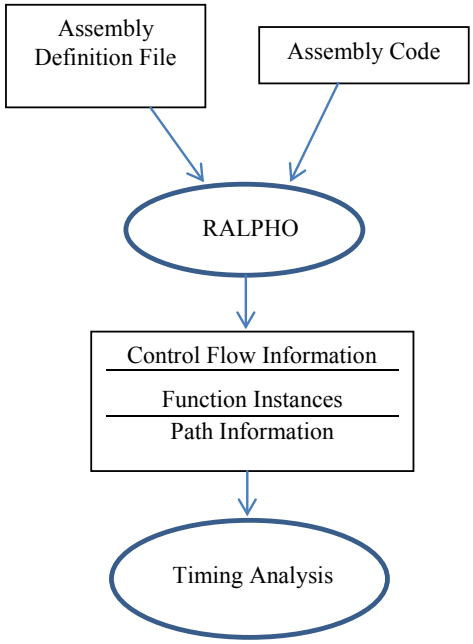


Fig. 1. Framework for RALPHO and timing analysis.

A. *RALPHO Preliminaries*

RALPHO first reads the ADF to understand the assembly syntax being used. The ADF contains the following information.

- Whether the architecture employs delayed branching
- Name of function call instruction
- Identifiers for code and data sections
- Pipeline stages
- Comment symbol
- Register names, types, and sizes
- Format of individual instructions in the instruction set, including number of cycles spent in each pipeline stage
- Branch and jump instructions

Next, RALPHO reads the assembly code. It aims to determine the overall structure of the source program. The highest level of program organization is the function. Since library functions are not evaluated by the timing analyzer, it is necessary for RALPHO to distinguish library functions from functions that are actually present in the program being analyzed. For each function call, it is necessary to determine if the function being called actually resides in the input source code, or is a function in the run-time library which is outside the scope of what is going to be analyzed by the timing analyzer. For each function, RALPHO detects its instructions and their operands. Then it proceeds to compute the function's basic blocks and loops, which are critically important for timing analysis.

Often, assembly code has labels to identify the targets of branch, jump and function call instructions. RALPHO determines which labels represent the start of a function, as opposed to the targets of ordinary branch and jump instructions. In some cases, branch and jump labels are not present in the assembly code. For example, the compiler may use the binary address instead as the destination of a branch or jump. In this case, RALPHO simply inserts an explicit label at the target instruction.

Next, RALPHO groups the instructions in each function into blocks. For the purpose of our analysis, a "block" is a basic block [1], which is a set of assembly instructions that are guaranteed to be executed together. In other words, once a block is entered, all of the remaining instructions in that block must execute. Blocks are delimited in two ways. First, a label, which usually represents the target of a branch, jump, or function call, signals the start of a block. Any transfer-of-control instruction signals the end of a block. RALPHO uses these features to delimit blocks.

Once the blocks have been determined, RALPHO next detects all of the loops in each function. This analysis begins by finding the successors, predecessors, and dominators of each block [1]. The dominators of a block B comprise those blocks where control must have traveled through in order to reach B. We identify which blocks comprise the loop, and which block is the loop header (i.e. the first block to be executed in the loop). We also determine if this loop is nested inside another loop, and the nesting level of the loop. This is important for loop iteration calculation, because the inner loop control variable may depend on an outer loop control variable [4]. To calculate the number of loop iterations, RALPHO finds the register representing the loop control variable, and then detects how this register is initialized before the loop, as well as the limit and the stride. A formula for computing the loop iterations is then used [4, 6].

B. *Branch Constraint Analysis*

After determining the basic structure of the source program, RALPHO looks at each loop in the program to predict branch behavior or correlation, which will be useful later in determining which paths are infeasible. The algorithm for determining infeasible paths is based on previous work [6]. Branch behavior is analyzed by two routines in RALPHO: compare_branches and check_loops.

Branches of a loop or function are considered in pairs. The function compare_branches returns a list of 4 elements. These indicate which of the 4 possible paths through the two branches is/are possible. These paths are: both branches taken, first one taken and second one falls through, first one falls through and second one taken, and both branches fall through. The routine

then determines which of these four scenarios is possible, and returns these answers to check_loops.

Compare_branches examines two branches on the same register within the same function. It considers 6*6*3 = 108 possible cases of branch correlation. The two branches being compared are of the form (R_i $relop_i$ $const_i$) and (R_j $relop_j$ $const_j$), where R_i and R_j are registers, $relop_i$ and $relop_j$ are each one of six possible relational comparisons (i.e. <, >, <=, >=, ==, !=), and $const_i$ and $const_j$ are the immediate values the registers are being compared to. There are 6 relational operators for each branch comparison. Each branch is comparing a register to a constant. The factor of 3 comes about due to the relationship between $const_i$ and $const_j$, namely that $const_i == const_j$, $const_i > const_j$, or $const_i < const_j$.

The routine check_loops involves these steps:

- First, for each register, we calculate in which blocks it is used in a comparison for a conditional branch.
- Second, for each register R, if there is more than one branch that uses R, let B_i and B_j be the blocks containing these branches. We invoke compare_branches(B_i, B_j) to determine how the branches correlate. The result is stored in block B_i.
- Third, we perform the branch prediction. For each block B_i, if B_i contains an instruction that assigns a constant to a register R, and R is used in a comparison in block B_j (which could be the same as B_i), temporarily create a temporary synthetic beq instruction and invoke compare_branches(B_i, B_j).
- Finally, sort the branch constraint information by block so that the information can be used by the timing analyzer.

RALPHO detects nine possible branch constraints, listed below. These constraints can be grouped into two categories. The first three deal with branch prediction, while the other six deal with branch correlation [7]. In each scenario below, it is assumed that block B contains a conditional branch that is taken or not; i.e. it branches or it falls through. In the second group of constraints, block A also has a branch instruction.

- Branch prediction constraints
  - Block A makes block B unknown
  - Block A makes block B fall through
  - Block A makes block B branch
- Branch correlation constraints
  - If A falls through, then B is unknown
  - If A branches, then B is unknown
  - If A falls through, B will branch
  - If A falls through, B will fall through
  - If A branches, B will branch
  - If A branches, B will fall through

Some of the branch states use the word “unknown” in place of fall through or branch. Unknown means that the effect of the current block A on the branch in block B is ambiguous or cannot be determined. In other words, there is not enough information to say definitively whether B will be taken or not. Marking a branch effect as unknown avoids the problem of a branch misprediction. As an example, consider the comparison i < 10. A block performing i++ would make the result of the comparison unknown. In most cases, incrementing i by 1 would have no effect on the branch, but in the case that i is being incremented from 9 to 10 would cause a change in the branch behavior. If both cases are plausible, RALPHO declares the branch to be unknown. Furthermore, in the current implementation of RALPHO, we only track branch correlations if the comparisons are made to constants. Statements comparing two registers, which could result from high-level statements such as x == y, are not handled at present, and these branches are treated as unknown. The effect of having an “unknown” branch is that the timing analyzer would not be able to rule out a path containing this branch even if it might be infeasible.

Finally, after all of the branch constraint information has been calculated, we wish to detect infeasible paths in each loop and function, and remove them from the analysis. For simplicity, a function is treated as a loop with one iteration. A loop consists of one or more blocks. Each block is annotated based on what effect it has on branches that occur throughout the loop. Later, paths are created, and then those containing infeasible block-to-block transitions are immediately removed.

III. Experimental Evaluation

We implemented the RALPHO system for gaining timing data for tasks running on an ARM7 processor. The development board we used was the NXP LPC2138-based Keil MCB-2140. A similar platform, the LPC2138, was used in the WCET Tool Challenge, 2011 [9]. The simulator we are using for testing the timing analyzer’s WCET estimates was Keil’s MDK-ARM Professional simulator for the ARM7.

Here is a simple illustrative example given to RALPHO and the timing analyzer. Suppose a function (which could also be a loop body) contains two if-statements as follows. In this example, after compilation, the code comprises five basic blocks, containing 5, 9, 2, 9 and 2 instructions, respectively.

```
// Block 1:  initial code in function

if (a == 1) {
   // Block 2:  body of if-statement
}

// Block 3:  up through next if-statement
// The variable a is not modified.

if (a != 1) {
   // Block 4:  body of if-statement
}

// Block 5:  rest of function
```

RALPHO is able to determine the following facts about the branch correlations. Blocks 1 and 3 have a relationship where if block 1 branches then block 3 falls through, and if block 1 falls through then block 3 must branch. RALPHO also detects that this relationship works in reverse as well: if block 3 branches (falls through), then block 1 falls through (branches). However, it turns out that no execution path encounters block 3 before block 1. Here are the possible execution paths based only on the control-flow of the blocks using predecessor and successor information:

Path 1 comprises blocks 1, 2, 3, 4, 5 and executes 27 instructions.

Path 2 comprises blocks 1, 2, 3, 5 and executes 18 instructions.

Path 3 comprises blocks 1, 3, 4, 5 and executes 18 instructions.

Path 4 comprises blocks 1, 3, 5 and executes 9 instructions.

When we also perform the branch correlation analysis in this example, the timing analyzer can automatically determine that paths 1 and 4 are infeasible. It turns out that one of the infeasible paths would have been the worst-case path if the static analysis did not consider branch constraints. As a result, the WCET can be exactly determined in this example, instead of overestimating the WCET by more than 30 percent. Incidentally, the best-case execution time (BCET) estimation is tightened as well because the shortest path has been eliminated from the timing analysis.

IV. Conclusion and Ongoing Work

The goal of this research is to provide a timing analysis tool with the control-flow and branch constraint information it needs to detect infeasible paths. What is novel about the approach is that we efficiently produce constraint information in RALPHO directly from the assembly code. In other words, it is no longer necessary to modify the back end of a compiler to produce this information [2, 7]. As a result, it is easier to retarget the analysis tool to a new architecture. Ongoing work includes testing our system with standard real-time benchmarks used in the WCET community.

This work continues in a couple of areas. We are currently modifying RALPHO to consider branch correlations where a register is compared to another register, instead of just a constant. Next, additional analysis can detect other value-dependent constraints, such as if a branch is taken on certain iterations of the loop. This can assist with analysis questions besides WCET [9], such as bounding the minimum or maximum number of times a certain path is taken or how many times a function is called. Finally, interprocedural (i.e. inter-function) analysis can be performed. For example, consider a loop containing a function call, and the called function itself contains a loop. It would be useful to be able to detect this implicit nested loop, and doing so can improve estimates of execution time.

References

[1] A. Aho, R. Sethi, J. Ullman, Compilers: Principles, Techniques, and Tools, Addison-Wesley, 1986.

[2] M. Benitez, J. Davidson, "A portable global optimizer and linker," Symposium on Programming Language Design and Implementation, 1988, pp. 77-98.

[3] B. Blackham, M. Liffiton, G. Heiser, "Trickle: automated infeasible path detection using all minimal unsatisfiable subsets," Real-Time and Embedded Applications Symposium, 2014.

[4] J. Coffman, C. Healy, F. Mueller, D. Whalley, "Generalizing parametric timing analysis," Proceedings of the ACM SIGPLAN Conference on Languages, Compilers and Tools for Embedded Systems, 2007, pp. 152-154.

[5] J. Estep, C. Healy, "A flexible tool for visualizing assembly code," Journal of Computing Sciences in Colleges, February 2005, pp. 55-67.

[6] C. Healy, M. Sjodin, V. Rustagi, D. Whalley, R. van Engelen, "Supporting timing analysis by automatic bounding of loop iterations," Real-Time Systems, May 2000, pp. 129-156.

[7] C. Healy and D. Whalley, "Automatic detection and exploitation of branch constraints for timing analysis," IEEE Transactions on Software Engineering, August 2002, pp. 763-781.

[8] P. Keim, A. Noyes, A. Ferguson, J. Neal, C. Healy, "Extending the path analysis technique to obtain a soft WCET," International Workshop on Worst-Case Execution Time Analysis, 2009, pp. 134-142.

[9] R. von Hanxleden et al., "WCET tool challenge 2011: report," International Workshop on Worst-Case Execution Time Analysis, 2011, pp. 104-138.

[10] R. Wilhelm et al., "The worst-case execution time problem – overview of methods and survey of tools," ACM Trans. Embed. Comput. Syst., 7(3):1-53, 2008.