A Filtering Approach to Distributed Priority Assignment in Real-Time Systems

Moritz Neukirchner, Rolf Ernst
Institut für Datentechnik und Kommunikationsnetze
Technische Universität Braunschweig
Email: neukirchner|ernst@ida.ing.tu-bs.de

***Abstract*—Recent advances in in-system performance analysis allow to determine feasibility of a system configuration within the system itself. Such methods have been successfully used to perform admission control for updates in distributed real-time systems. Parameter synthesis, which is necessary to complement the admission control with self-configuration capabilities, lacks behind because current approaches cannot be distributed properly or because of necessary design-time preprocessing steps.**

In this paper we present a novel distributed algorithm to find feasible execution priorities in distributed static-priority-preemptively (SPP) scheduled real-time systems under consideration of end-to-end path latencies. The presented algorithm builds on top of an existing distributed feasibility test, which is derived from compositional performance analysis [1]. With an extensive set of pseudo-randomly generated testcases we demonstrate the applicability of the approach and show that the proposed algorithm can even compete with state-of-the-art design time tools at a fraction of the runtime.

I. Introduction

The integration of several components to a complex real-time system, such as an automotive platform, is a challenging task, as the integration process may introduce non-functional dependencies among otherwise independent components. Such dependencies arise through e.g. use of a common communication bus. Correct functioning is usually assured at design-time through means of extensive testing and formal verification. This has to be performed for every possible system configuration/variant. This approach becomes infeasible if user-driven software updates are allowed - as anticipated even for automotive system [2], [3] - because future system configurations become unpredictable.

This problem can be addressed by admission control mechanisms, that perform a formal verification in the system itself [4]. In this case only every actually encountered configuration is analyzed prior to system reconfiguration/update and infeasible changes are rejected.

However, in real-time systems feasibility of a system configuration heavily depends on the assignment of scheduling parameters. Thus, when updating a system, by e.g. adding a new software component, the update may be deemed infeasible by the admission control, although it may be feasible under a different assignment of scheduling parameters. To address this issue, we propose to extend admission control by a self-configuration service, which reassigns scheduling parameters such that configurations, that would otherwise be rejected, can be allowed to execute.

Specifically, we address the constraint satisfaction problem (CSP) of finding feasible priority assignments in SPP scheduled real-time systems under consideration of end-to-end path latency constraints. The algorithm that we present relies on a distributed implementation of compositional performance analysis [5], [1], which has been successfully used for a distributed admission control scheme [4]. In order to reduce runtime overhead and to integrate with the admission control scheme the proposed algorithm is also implemented distributedly. Despite its application to the above admission control scheme, the approach is generally applicable to the problem of distributed priority assignment and not bound to any specific framework. As we show later, the algorithm can even outperform a state of the art design time tool.

The remainder of this work is structured as follows. First we will review previous approaches for priority assignment in real-time systems (section II). We will then provide a brief description of the system model and provide some insight on the underlying admission control scheme [4] and its distributed feasibility test [1] (section III). In section IV we will outline the general strategy for the distributed self-configuration algorithm as presented in [21]. Section V then describes our novel algorithm for distributed priority assignment, which integrates into this general strategy. We evaluate its performance with two benchmark algorithms (section VI). Section VII concludes the paper.

II. Related Work

The problem of priority assignment has been studied intensively in the scope of scheduling analysis. First approaches addressed uni-processor systems and the question of schedulability of periodic tasks with task deadlines equal to their periods [6]. Later work reduced the restrictions on task deadlines [7], [8] and on periodicity [9], [10]. Extensions to multi-processor systems were then presented in [11], [12], [13]. In our scenario of in-field updates we consider tasks with communication dependencies and constraints on end-to-end path latencies. As the above approaches consider independent tasks and task- rather than path-latencies they are of limited applicability.

[14] and [15] both presented frameworks for design-space exploration of real-time systems that do not pose these restrictions on the system model. Both approaches use a genetic

algorithm (GA) and a tool for performance analysis [5] to explore the design space. They support a multitude of parameters for optimization, among which is priority assignment. In this work we will use [14] as a benchmark. Genetic algorithms are generally computationally expensive due to the large number of individuals that are required to derive a solution. This is undesirable in resource-constrained embedded systems.

Another approach that is specifically targeted at runtime assignment of scheduling parameters was presented by [16], [17]. Here, a control-theoretic approach is taken to dynamically adjust scheduling parameters based on the actual workload of the system. This approach, however, is only suitable for soft real-time systems and cannot be applied if hard constraints have to be considered.

An approach that is more suitable for use in in-system admission control, that shall ensure adherence to hard constraints, is to divide end-to-end deadlines into local deadlines. Based on local algorithms tasks are then scheduled w.r.t. their local deadlines. [18] provides a good overview of work following this approach. While most of these approaches target design-time optimization, e.g. [19], the algorithm presented in [20] aims to find feasible schedules in-system. However, the calculation of the local deadlines has to be performed in an offline pre-processing step, which can significantly limit the exploitation of available system slack.

[21] presented a distributed heuristic priority assignment algorithm, that does not require division of path latency deadlines into local task deadlines, while still allowing an efficient distributed implementation. The algorithm presented in this paper builds on the same distribution approach as [21] (see section IV). However, as will be shown in section VI, our algorithm provides greatly improved results and even outperforms the design-time solution [14], which was chosen as benchmark.

III. System Model & Admission Control concept

In this section, we introduce the system model and admission control concept, which forms the basis for our algorithm.

We use the system model as in [5]. In this system model a hardware *platform* $\mathcal{P}$ consists of multiple processors interconnected by communication media. We will refer to processors and communication media as (computational and communication) *resources* ρ_j. On this platform a set of potentially communicating *tasks* $\Gamma = \{\tau_i\}$ are executed. A set of paths $\Psi = \{\psi_k\}$ with constraints on end-to-end latencies $C = \{\chi_{\psi_k} : \psi_k \in \Psi\}$ are specified for the task set.

Additionally, we assume that the distributed performance analysis (DPA) algorithm as presented in [1] is running on this platform to perform admission control [4]. This DPA algorithm follows the general approach of compositional performance analysis [22], [5], which composes local schedulability analysis algorithms using event model interfaces. Schedulability analysis algorithms derive worst-case response times from worst-case execution times of tasks for a given scheduling policy. Algorithms exist for a multitude of scheduling and bus arbitration schemes, e.g. for static priority preemptive scheduling [23], Round Robin [24], or CAN Bus [25]. Using these functions, Compositional Performance Analysis derives bounds on the individual response time of each task in the system also under the assumption of communicating tasks. The response times are aggregated to compute bounds on path latencies [26].

The distributed implementation is composed of several DPA instances, one residing at reach resource in the system. The single instances communicate the worst-case timing behavior of their tasks and cooperatively determine worst-case system level timing. Each DPA instance only contains model data of tasks that reside on the resource of that DPA. We refer to this information as the *local model* of the DPA instance. Specifically, a DPA instance can provide information on worst-case task response times ω_{τ_i} (WCRT), worst-case path latencies λ_{ψ_k} and path latency constraints $\chi_{\psi_k} \in C$ of tasks that reside on the same resource as the DPA instance. The provided estimations on WCRTs are *monotonic*, i.e. if a task's priority is increased/decreased and all other parameters remain equal, its worst-case response time can only decrease/increase or remain equal, respectively.

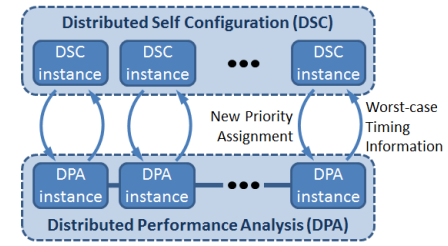
To be able to reason about data within the local model of a DPA instance we introduce some sets of variables. Let Γ_{ρ_j} be the set of tasks that are mapped on resource ρ_j and Γ_{ψ_k} be the set of tasks that are part of path ψ_k. Furthermore let Ψ_{ρ_j} be the set of paths that have at least one task mapped on resource ρ_j ($\tau_i \in \Gamma_{\rho_j}$) and Ψ_{τ_i} be the set of paths that task τ_i is part of ($\tau_i \in \Gamma_{\psi_k}$). These definitions are later required to define the self-configuration algorithm.

IV. Self-Configuration Strategy

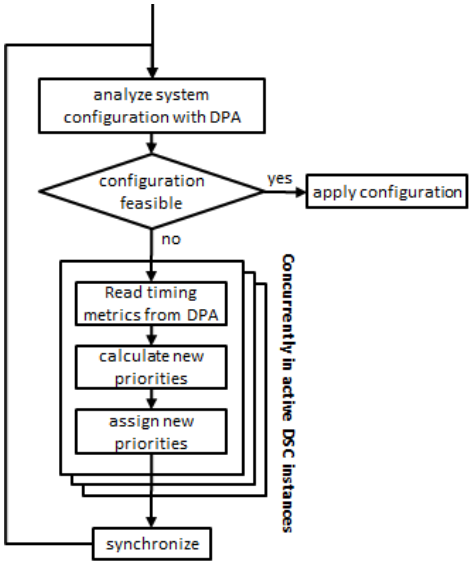
We employ the general approach presented in [21], which we outline in this section.

The distributed self-configuration (DSC) algorithm relies on the model-based Distributed Performance Analysis (DPA) algorithm [1] which is used for admission control. Each DPA instance is complemented by a DSC instance - both instances residing at the same resource (figure 1a). The DSC can request estimations on WCRTs and path latencies from the DPA. Due to the distribution of the model and the DPA each DSC instance can only access the information provided by its attached DPA instance, i.e. the DPA's local model. Each DSC instance can reassign task priorities in the local model of its attached DPA instance. While the DPA instances communicate to analyze a system configuration, the DSC instances do not require to communicate except for synchronization.

Figure 1b shows the DSC flow. The DPA analyzes the system model. A DSC instance becomes active when its DPA instance detects an infeasible update to the system configuration, if the worst-case path latency of any path on that resource exceeds its constraint. Based on local rules and data available from their attached DPA instances all active DSC instances concurrently compute new priority assignments and insert them into the model of the DPA. All active DSC instances synchronize (e.g. using a barrier synchronization protocol as described in [27]) to ensure a consistent model.



(a) Architecture view



(b) Logic view

Fig. 1: General algorithm flow

Then the DPA analyzes the modified configuration again. This loop is executed on a resource whenever the current priority assignment does not satisfy all path latency constraints. Each execution of this loop is referred to as a *DSC step*. As the DPA is performed synchronized across all affected resources, DPA and DSC are performed in a lock-step manner. If a global solution is found (i.e. all constraints are satisfied), a feasible configuration has been found and the update to the system configuration can be accepted. To avoid endless loops in case of unsatisfiable constraints in an update, the number of DSC steps can be supervised and bounded by an additional software component. All computation and communication of DPA and DSC can be performed on lowest priority, to minimize the effect on running applications.

The algorithm of [21] within each DSC is based on a metric that indicates the "responsibility" of task for a path latency constraint violation - the local improvement target (LIT). [21] formally defines it as

Definition 1: Let the local improvement target δ_{τ_i} of task $\tau_i \in \Gamma_{\rho_j}$ be defined as

$$\delta_{\tau_i} = \max_{\psi_k \in \Psi_{\tau_i}} \left(0, \frac{\omega_{\tau_i}}{\lambda_{\psi_k}} * (\lambda_{\psi_k} - \chi_{\psi_k})\right) \quad (1)$$

The LIT is the maximum quotient of the task's WCRT and the path latency multiplied by the path violation. The maximum is taken over all paths of the task. The calculation only requires the task's worst-case response times ω_{τ_i} and path latencies λ_{ψ_k} and latency constraints χ_{ψ_k} of all paths $\psi_k \in \Psi_{\tau_i}$, that the task is part of. All of this information is provided by the DPA based on its local model. Thus LITs can be calculated at DSC instances without the necessity of explicit communication among the different DSC instances.

[21] noted, that a self-configuration that distributedly assigns task priorities in decreasing order of their LIT exhibits oscillatory behavior, thus leading to poor search space coverage and poor algorithm performance. The authors propose to randomly inhibit the execution of single DSC instances according to a "lazy threshold" to break the oscillatory loops. This leads to greatly improved results.

In the following section we introduce a novel strategy to break the oscillatory loops. Section VI will show that this new approach provides even larger improvements and yields results comparable to those of centralized design-time tools.

V. Distributed Self-Configuration Algorithm

We propose to use a control-theory inspired approach within each DSC instance to assign priorities without causing oscillatory loops. Note, that although assignment of task priorities is described, communication between tasks over physical communication media is assumed to be scheduled priority-based as well and thus can be handled in the same fashion.

[21] aimed to reduce oscillations by introducing random pertubations into the self-configuration process. We propose to address the issue of oscillations systematically by considering past DSC steps in the priority assignment process. However, logging all evaluated configurations of previous DSC steps is intractable because 1. no DSC instance has a complete view of the system model and thus cannot decide alone whether a configuration has already been evaluated 2. logging all previous configurations introduces significant memory overhead, which may be prohibitive, if the algorithm is used in-system along with an admission control scheme.

We propose a novel DSC algorithm that complements the priority assignment in decreasing order of the LIT with a time-discrete PID filter, as depicted in figure 2. Such a filter allows to track past DSC steps with minimal memory overhead. In a first step (1. in fig. 2) *set point priorities* S_{τ_i} are assigned in decreasing order of the tasks' LIT. This is the priority assignment, which tends to oscillate if no further countermeasures are taken. These set point priorities are input to a feedback PID-controller (2. in fig. 2). This filter returns a *priority rating* R_{τ_i} which incorporates the set-point priorities S_{τ_i}, the currently assigned priorities P_{τ_i} and the history of DSC steps through a proportional (P), an integral (I) and a derivative (D) component. The proportional component is equivalent to a scalable priority assignment in direct correleation with the LIT. The integral component allows to super-proportionally increase the priority rating of a task if it violates any of its path latency constraints over several subsequent DSC steps. The derivative component damps this effect by decreasing the priority rating if it increased in the previous DSC step.

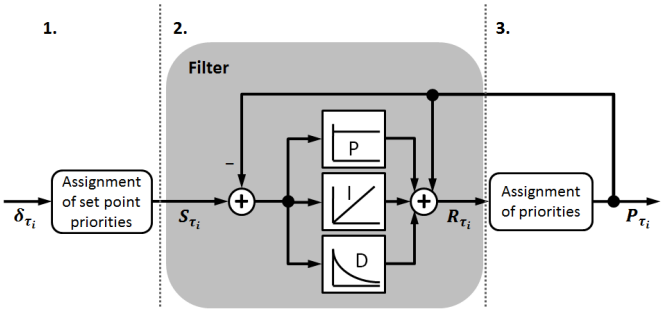


Fig. 2: Feedback control for priority assignment

In combination all three components allow to calculate a sufficiently stable priority rating. Once the priority ratings have been calculated tasks are assigned priorities in decreasing order of these ratings (3. in fig. 2).

The described flow is further detailed in algorithm 1, which shows the l-th DSC step. In a first step all local improvement targets are calculated (line 3). To assign the set point priorities for each task $\tau_i \in \Gamma_{\rho_j}$, Γ_{ρ_j} is sorted in descending order of the LITs δ_{τ_i} (line 6). The set point priority of a task τ_i is then set to its position in the sorted Γ_{ρ_j} (line 7).

The set point priorities S serve as input for the filter. Let $\Delta_{\tau_i}(l)$ be the difference of assigned priority and set point priority in the l-th DSC step, i.e.

$$\Delta_{\tau_i}(l) = S_{\tau_i}(l) - P_{\tau_i}(l-1) \tag{2}$$

The priority rating $R_{\tau_i}(l)$ in the l-th DSC step is then calculated by

$$\begin{aligned} R_{\tau_i}(l) = \quad & P_{\tau_i}(l-1) \\ & + k_P * \Delta_{\tau_i}(l) \\ & + k_I * I_{\tau_i}(l) \\ & + k_D * D_{\tau_i}(l) \end{aligned} \tag{3}$$

with

$$I_{\tau_i}(l) = I_{\tau_i}(l-1) + \Delta_{\tau_i}(l) \tag{4}$$

$$D_{\tau_i}(l) = \Delta_{\tau_i}(l-1) - \Delta_{\tau_i}(l) \tag{5}$$

and

$$k_P, k_I, k_D \in \mathbb{R} \tag{6}$$

The parameters k_P, k_I and k_D are the gain parameters of the proportional, integral and differential components of the filter, respectively. After calculation of the priority ratings $R_{\tau_i}(l)$ for all $\tau_i \in \Gamma_{\rho_j}$ (line 9), the set Γ_{ρ_j} is sorted in descending order of the priority ratings $R_{\tau_i}(l)$ (line 11). The priority of all tasks $\tau_i \in \Gamma_{\rho_j}$ is then set to their respective position in the sorted Γ_{ρ_j} (line 12).

Note that the addition of the filter does not prevent oscillations from ever occurring. The inputs of each DSC instance depend on the behavior of potentially several other DSC instances - while these dependencies change with different priority assignments. Thus, stability of the constraint solving process cannot be guaranteed. Furthermore, as the feedback

Algorithm 1 l-th DSC Step (filtered LIT-based)

1: **for** $\rho_j \in \mathcal{P}_s \subseteq \mathcal{P}$ **concurrently do**
2: **for** $\tau_i \in \Gamma_{\rho_j}$ **do**
3: calculate δ_{τ_i}
4: **end for**
5: **if** any $\delta_{\tau_i} > 0 : \tau_i \in \Gamma_{\rho_j}$ **then**
6: sort Γ_{ρ_j} descending in δ_{τ_i}
7: assign set point priorities in order of sorted Γ_{ρ_j}
8: **for** $\tau_i \in \Gamma_{\rho_j}$ **do**
9: calculate $R_{\tau_i}(l)$
10: **end for**
11: sort Γ_{ρ_j} descending in $R_{\tau_i}(l)$
12: assign priorities in order of sorted Γ_{ρ_j}
13: **end if**
14: **end for**

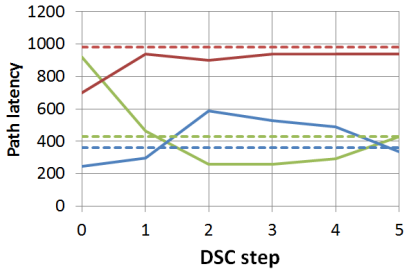


Fig. 3: Example: Path latency over several DSC steps

path of the filter includes a sorting operation, it is intractable to calculate optimal gain parameters to achieve a certain damping. Instead we have determined suitable gain parameters (equations 7-9) empirically on the testcases that were used in the evaluation (section VI).

In order to find suitable gain parameters we have observed the change of path latencies of several testcase systems over the course of several DSC steps. Figure 3 shows a plot of this for one testcase system, which consists of 4 resources, 10 tasks and 5 communication channels on one communication medium. Three paths with constrained latency are defined. The plot shows the path latencies in solid lines and the respective constraint in the same color as dashed line. We see that the experimentally chosen gain parameters cause a decent settling behavior of the path latencies towards their respective contraints, rendering the system feasible after 5 DSC steps. The gain parameters that were used for this testcase as well as the evaluation in section VI are given below.

$$k_P = -0,4 \tag{7}$$

$$k_I = 0,05 \tag{8}$$

$$k_D = -0,1 \tag{9}$$

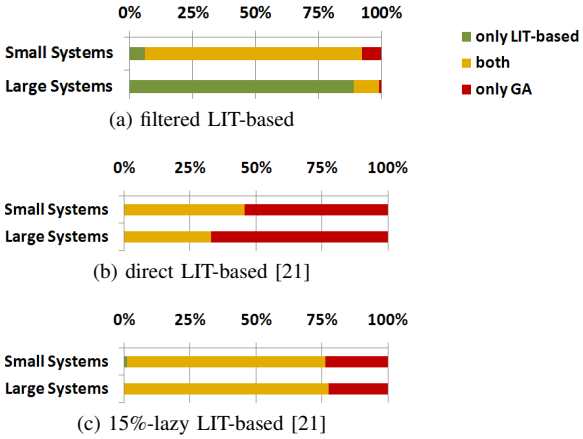
Thus, although we cannot guarantee optimality of the filter we observe, that the possibility to rank tasks based on their value-continuous priority rating - instead of the stepwise chaning path latency - and because these priority ratings incorporate the history of DSC steps, a more stable trend for priority assignment is obtained. In the following section we show on a larger set of testcases that this filtering approach indeed poses a suitable approach to priority assignment.

VI. Evaluation

In this section we evaluate the performance of the proposed algorithm. As it employs a heuristic and does not necessarily find a feasible configuration we have tested it on an extensive set of testcases. As baseline for the comparison we use a state of the art design-time tool, which is based on a genetic algorithm [14]. Furthermore, we compare the performance to the lazy algorithm [21], which builds on the same general DSC approach as this paper while reducing oscillations by means of a lazy threshold.

The testcase systems were generated with the open-source tool *System Models for Free* (SMFF) [28], [29], which pseudo-ramdomly generates completely specified system models. We have used two different parameter sets to evaluate scalability of the approach. The first parameter set generates smaller systems with 4 computational resources, 2 to 3 communication resources and 2 to 4 tasks per task set. The second parameter set generates system models with 12 computational resources, 3 to 5 communication resources and 3 to 7 tasks per task set. The number of task sets per testcase depends on the success of the filtered LIT-based algorithm and the GA. We have added additional task sets until the system turned infeasible. Then we have used both algorithms to find a feasible configuration. If one succeeded, we added another task set. This was repeated until neither algorithm was able to find a feasible configuration. As a result the different testcases contained between 2 and 10 task sets (i.e. in total 4-70 tasks) and in total 118 testcases were analyzed per paramter set. For both parameter sets the worst-case execution times of each task were set such that it caused a load (i.e. WCET/Period) between 1% and 5%. Constraints on end-to-end path latency were set to values 3 to 5 times larger than the sum of the WCETs of all tasks along the path. For reproducibility of results we provide the complete set of parameters in figure 6.

As a first metric for evaluation we use the number of false negatives of each algorithm. In most cases it is intractable, to analyze whether a system has a feasible priority assignment at all, as the number of possible configurations easily reaches values greater than several 100 million. As a consequence we will call an optimization run a *false negative* for the filtered LIT-based algorithm/GA, if the filtered algorithm/GA failed to find a solution, while the other did find a feasible priority assignment, respectively. The LIT-based algorithm was restricted to 500 DCS steps, while the GA analyzed 50 individuals over 10 generations. Figure 4a shows the percentage of testcases where only the filtered LIT-based algorithm (green), both algorithms (yellow) and only the GA (red) found a solution. For the small systems the novel filtered LIT-based algorithm and the genetic algorithm [14] performed equally well. In more than 80% of the testcases both algorithms found a solution. In ∼8% of the testsystems each algorithm was able to find a solution while the other failed to find a feasible priority assignment. From the larger testcase systems we see that the filtered algorithm scales significantly better than the GA. In ∼90% of the large testcase systems only the filtered



(a) filtered LIT-based

(b) direct LIT-based [21]

(c) 15%-lazy LIT-based [21]

Fig. 4: Comparison w.r.t. solved testcases

algorithm was able to find a feasible priority assignment while the GA failed to find a solution. Conversely, in only ∼1% of the larger testcases the GA found a solution while our novel approach failed to solve the priority assignment problem. As we have used the same test setup as [21] we can also compare the performance of the filtered algorithm to that of the priority assignment in direct order of the LITs (figure 4b) and to the lazy LIT-based algorithm (figure 4c). Comparing the results to figure 4a we see that the filtered approach performs significantly better than either of the previous approaches.

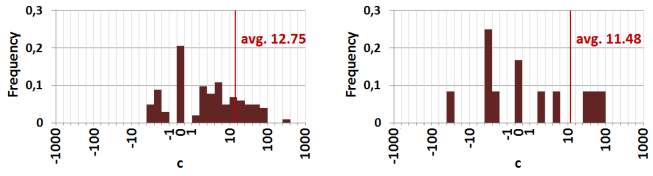
Next we compare the runtime of the algorithms. As the runtime of the self-configuration algorithm is dominated by the underlying DPA we use the number of required performance analysis runs (i.e. number of DSC steps) to obtain a solution as runtime metric. We introduce the comparison factor c, that signifies the relation of the number of analysis runs needed by the GA b and the number of analysis runs needed by the distributed LIT-based algorithm a.

$$c = \begin{cases} a/b & \text{if } a > b \\ 0 & \text{if } a = b \\ -b/a & \text{if } b > a \end{cases} \tag{10}$$

For both parameter sets figure 5 shows histograms over the comparison factor c for those testcases where both algorithms found a solution. Regardless of the testcase size the proposed LIT-based algorithm requires an order of magnitude less performance analyses to derive a feasible priority assignment than the GA, which is a state of the art design time tool (12.75x and 11.48x average improvement for the small and large testcase systems, respectively). For the same set of testcase generation parameters the lazy LIT-based algorithm of [21] states average runtime improvements vs. the GA of 5x and 7x for the two testcase parameter set. Thus, the novel filtered algorithm is ∼2x faster than the lazy algorithm while it is able to solve significantly more testcases.

VII. Conclusion

In this paper we have presented a novel algorithm that distributedly finds feasible priority assignments in distributed



(a) filtered LIT-based, Param. Set 1 (b) filtered LIT-based, Param. Set 2

Fig. 5: Improvement histograms

SPP scheduled systems under consideration of end-to-end path latency constraints. Due to the possibility of distributed implementation the algorithm can be used to complement an admission control scheme as in [4] to enhance a system by self-configuration capabilities.

In an evaluation based on an extensive set of pseudo-randomly generated testcases we have shown that the proposed filtered LIT-based algorithm was able to solve as many small testcases systems as a current design-time tool. For larger more complex systems it even outperformed the existing software significantly. At the same time, irrespective of the testcase size the proposed algorithm was more than an order of magnitude faster than the design-time tool.

Thus, although designed for an in-system distributed implementation, the algorithm poses an attractive choice for design-time configuration synthesis.

REFERENCES

[1] S. Stein, A. Hamann, and R. Ernst, "Real-time property verification in organic computing systems," in *Second Int'l. Symp. on Leveraging Applications of Formal Methods, Verification and Validation*, 2006.

[2] C. Hammerschmidt, "Bosch sees massive challenges ahead of automotive electronics," EETimes, Internet, June 2010. [Online]. Available: http://eetimes.eu/en/bosch-sees-massive-challenges-ahead-for-automotive-electronics.html?cmp_id=7&news_id=222902475&vID=8

[3] J. Reed, "Audi to relaunch a2 city model with 'apps' for bespoke features," Financial Times, Internet, May 2010. [Online]. Available: http://www.ft.com/cms/s/0/c4f14878-6c4a-11df-86c5-00144feab49a.html

[4] M. Neukirchner, S. Stein, H. Schrom, and R. Ernst, "A software Update Service with Self-Protection Capabilities," in *Proc. of the conf. on Design, Automation and Test in Europe (DATE)*, 2010.

[5] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst, "System level performance analysis - the symta/s approach," *Computers and Digital Techniques, IEE Proc. -*, vol. 152, pp. 148–166, 2005.

[6] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. ACM*, vol. 20, pp. 46–61, 1973.

[7] J. Y. T. Leung and J. Whitehead, "On the complexity of fixed-priority scheduling of periodic, real-time tasks," *Performance Evaluation*, vol. 2, pp. 237–250, 1982.

[8] N. Audsley, A. Burns, M. F. Richardson, and A. J. Wellings, "Hard real-time scheduling: The deadline-monotonic approach," in *in Proc. IEEE Workshop on Real-Time Operating Systems and Software*, 1991.

[9] J. Lehoczky and S. Ramos-Thuel, "An optimal algorithm for scheduling soft-aperiodic tasks in fixed-priority preemptive systems," *Real-Time Systems Symp., 1992*, pp. 110 –123, 1992.

[10] R. Davis and A. Burns, "Optimal priority assignment for aperiodic tasks with firm deadlines in fixed priority pre-emptive systems," *Information Processing Letters*, vol. 53, pp. 249 – 254, 1995.

[11] M. Bertogna, M. Cirinei, and G. Lipari, "New schedulability tests for real-time task sets scheduled by deadline monotonic on multiprocessors," in *Principles of Distributed Systems*. Springer Berlin / Heidelberg, 2006.

[12] B. Andersson, "Global static-priority preemptive multiprocessor scheduling with utilization bound 38%," in *Principles of Distributed Systems*. Springer Berlin / Heidelberg, 2008.

[13] R. I. Davis and A. Burns, "Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems," in *30th IEEE Real-Time Systems Symp. (RTSS)*, 2009.

[14] A. Hamann, M. Jersak, K. Richter, and R. Ernst, "A framework for modular analysis and exploration of heterogeneous embedded systems," *Real-Time Syst.*, vol. 33, pp. 101–137, 2006.

[15] M. Glaß, M. Lukasiewycz, J. Teich, U. Bordoloi, and S. Chakraborty, "Designing heterogeneous ecu networks via compact architecture encoding and hybrid timing analysis," in *Proc. of the 2009 Design Automation Conference (DAC)*, 2009.

[16] L. Palopoli, L. Abeni, T. Cucinotta, G. Lipari, and S. Baruah, "Weighted feedback reclaiming for multimedia applications," in *IEEE/ACM/IFIP Workshop on Embedded Systems for Real-Time Multimedia (ESTImedia)*, 2008.

[17] T. Cucinotta and L. Palopoli, "QoS Control for Pipelines of Tasks using Multiple Resources," *IEEE Trans. on Computers*, vol. 59, pp. 416–430, 2010.

[18] J. Jonsson and K. G. Shin, "Robust adaptive metrics for deadline assignment in distributed hard real-time systems," *Real-Time Systems*, vol. 23, pp. 239–271, 2002.

[19] J. G. García and M. G. Harbour, "Optimized priority assignment for tasks and messages in distributed hard real-time systems," in *Proc. of the IEEE Workshop on Parallel and Distributed Real-Time Systems*, 1995.

[20] M. D. Natale and J. A. Stankovic, "Dynamic end-to-end guarantees in distributed real time systems," in *Real-Time Systems Symp.*, 1994.

[21] M. Neukirchner, S. Stein, and R. Ernst, "A lazy algorithm for distributed priority assignment in real-time systems," in *Proc. of 2nd IEEE Workshop on Self-Organizing Real-Time Systems (SORT)*, 2011.

[22] K. Richter, "Compositional scheduling analysis using standard event models," Ph.D. dissertation, Technical University of Braunschweig, Department of Electrical Engineering and Information Technology, 2004.

[23] K. W. Tindell, "An extendible approach for analysing fixed priority hard real-time systems," *Journal of Real-Time Systems*, vol. 6, pp. 133–152, 1994.

[24] R. Racu, L. Li, R. Henia, A. Hamann, and R. Ernst, "Improved response time analysis of tasks scheduled under preemptive round-robin," in *Proc. of the Int'l. Conf. on Hardware-Software Codesign and System Synthesis*, 2007.

[25] R. I. Davis, A. Burns, R. J. Bril, and J. J. Lukkien, "Controller area network (can) schedulability analysis: Refuted, revisited and revised," *Real-Time Syst.*, vol. 35, pp. 239–272, 2007.

[26] S. Schliecker and R. Ernst, "A recursive approach to end-to-end path latency computation in heterogeneous multiprocessor systems," in *Proc. 7th Int'l. Conf. on Hardware Software Codesign and System Synthesis (CODES-ISSS)*, 2009.

[27] S. Stein, M. Neukirchner, H. Schrom, and R. Ernst, "Consistency challenges in self-organizing distributed hard real-time systems," in *Proc. of IEEE Workshop on Self-Organizing Real-Time Systems - SORT 2010*, 2010.

[28] M. Neukirchner, S. Stein, and R. Ernst, "SMFF: System Models for Free," in *2nd Int'l. Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, 2011.

[29] M. Neukirchner, "System models for free (smff)," Internet, 2011. [Online]. Available: http://smff.sourceforge.net

APPENDIX

	BatchPlatformFactory			*StdTgffApplicationFactory*					*StdMapper*				*RandomPriorityAssigner*	*TaskLoadTimingFactory*					*LaxityConstraintFactor*		
		resCnt	meanCResPercentage		numTasks	diffNumTasks	tasMaxDegrIn	taskMaxDegrOut		kPredecessor	kSuccessor	kResDist	no parameters		minActPeriod	maxActPeriod	minTaskLoad	maxTaskLoad	bcetPercentage		minConstrFactor	maxConstrFactor
Test 1		3	0,35		3	1	1	1		3,0	3,0	1,5	n.a.		1000	2000	1%	5%	1,0		3,0	5,0
Test 2		12	0,35		5	2	1	1		3,0	3,0	1,5	n.a.		1000	2000	1%	5%	1,0		3,0	5,0

Fig. 6: Parameters for testcase generation