Distributed Performance Control in Organic Embedded Systems

Steffen Stein, Rolf Ernst

Institute of Computer and Communication Network Engineering
Technical University of Braunschweig, Germany
{stein | ernst}@ida.ing.tu-bs.de

Abstract. This paper introduces compositional performance analysis into evolving organic systems. It presents a layered distributed framework that can follow the platform and system evolution, continuously monitoring the effect of changes in the application on real-time constraints. For that purpose, an existing methodology based on iterative compositional performance analysis was adapted to a distributed algorithm. A buffering strategy is introduced to improve the algorithm convergence to the same order as the existing centralized offline algorithm. The effects are demonstrated in experiments.

1 Introduction

Organic Computing [10] has recently emerged as a new challenge in computer science. As ubiquitous and embedded computing systems become increasingly powerful, the development paradigms shift from implementing the technically possible to building robust and easily usable systems. Organic computing systems tackle this challenge by introducing adaptation, learning and self-configuration into complex computer systems. Initiatives as IBM's Autonomic Computing Initiative [8] or Intel's Proactive Computing [14] show that this is not only an academic endeavour.

Real-time systems constitute a notable share of todays embedded computers that needs special attention. The Design of robust and fault-tolerant real-time systems is a highly active research area, that has produced numerous approaches for evaluating and increasing system robustness against selected fault scenarios. Existing approaches use offline sensitivity analysis to optimize for robustness, meaning low sensitivity [6]. These methodologies can be applied throughout the design process of an embedded system and yield systems that are highly robust against a selected set of disturbances in the field.

Future embedded systems however, will undergo an evolution in both hard- and software configuration during their lifetime. In the automotive industry, it is already common to update or add software components during the lifetime of a product, producing a variety of software configurations in the field. To ensure functional and temporal correctness of all possible configurations, OEMs have to maintain a complex versioning database and perform exhaustive testing to cover the whole configuration landscape. This already constitutes a problem today, which will grow into a major challenge in the future. Designing embedded systems robust and fault-tolerant will not ultimately solve

this problem, as the evolution an embedded systems goes through during its lifetime cannot be foreseen at design time.

Introducing self-*-properties into embedded system will enhance them by flexibility for future updates in hard- or software, thus enabling evolution during their lifetime. Key properties of evolving (embedded) systems are the ability to assess its current situation (self-awareness) and to reconfigure themselves (self-configuration) in order to adapt to new situations as may be implied by software updates. For hard real-time systems, the challenge of implementing self-configuration and adaptation is not only to ensure functional, but also temporal correctness of a system.

This paper will introduce on a concise problem statement, highlighting the timing-related problems and challenges caused by the evolution of embedded real-time systems. It will then present a control framework building on formal methods for performance analysis capable of managing evolution while still ensuring temporal correctness of a system utilizing established methodologies from literature. It will close with an experimental examination of a prototype implementation the formal analysis engine.

The remainder of this paper is organized as follows. In the next section, we will introduce related work to then go into detail on the challenges to be addressed by our approach. The fourth section discusses the architecture of a performance control framework, where the fifth section goes into detail on the analysis methodology used by our framework. Before we conclude the paper, experimental results are presented.

2 Related Work

Designing adaptive, self-organizing real-time systems touches two highly active fields of current research. For once, we need to consider current development in the field of adaptive and fault-tolerant system design, but we also need to have a closer look into research concerning the analysis of timing properties of a given real-time system.

Currently, fault-tolerant and resilient systems are built by introducing explicit redundancy on a per computer level, such as TMR in avionics.

Later research has introduced redundancy not on a per computer, but on a per task level. The RecoNets project [7] has designed a prototype system consisting of multiple microcontrollers running a driver assistance application. It can survive failure of one or more board, since each task is shadowed on another microcontroller. Checkpointing techniques allow to seamlessly migrate execution of tasks from one microcontroller to another in case a failure is detected. This approach, however does not take global system timing issues into account when spawning shadow tasks at different points in the system.

Recently, also design of robust and fault tolerant systems taking into account timing properties has been tackled. In the AiS project [5] for example, common fault scenarios are identified and analysed for their possible impact on system performance. For selected scenarios, compensation mechanisms are implemented in the system, making it robust against these faults.

In the context of the “Organic Computing” priority program of the German DFG [10], many projects aim at building adaptive and self-configuring systems. This is usually achieved by extending the system by a control loop that observes the current sys-

tem state, evaluates it and performs control operations based on a knowledge base that may be constructed using reinforcement learning techniques such as Learning Classifier Systems [1]. These architectures are referred to as Observer/Controller Architectures; a general discussion of which can be found in [4].

In addition to current research in fault-tolerant and resilient systems, research in formal performance analysis of real-time systems has to be considered. In the past years, several approaches to system level timing analysis have been proposed by different research groups (i.e. [11, 2, 15, 12]). System level timing analysis requires task-level worst-case execution times as input data. Recent research has produced formal approaches to derive these from a given task description [16].

For the purpose of this paper, we can divide the approaches to system level performance analysis in two classes. Holistic approaches that try to use as much information as possible in order to perform a tight analysis of the real-time behaviour of a given system and compositional approaches that are capable of making abstractions at intermediate analysis steps.

The first class of techniques yields tightly bounded results on the timing properties of a given system at the prize of high computational complexity. Current approaches use different semantics to describe their systems ranging from dataflow graphs [11] to timed automata [9].

The second class, like the approaches proposed in [15] or [12] trade analysis accuracy for computational complexity. Here, local analysis techniques are composed using load descriptions of intermediate event streams.

Dynamic scheduling algorithms (i.e. [3]) adapt the scheduling parameters to a change in load conditions. Global schedulers can cover several processors, but only following a coherent homogeneous scheduling approach. In this sense, they are comparable to holistic analysis approaches. Global scheduling algorithms also do not easily adapt to changing hardware topologies and timing constraint types. Furthermore, they do not take system properties such as end-to-end latencies into account.

3 Problem formulation

For the means of this paper, we focus on loosely coupled distributed real-time systems as can be found i.e. in cars. A real-time system can generally be described as a set of processing units (processor, PU) interconnected by busses, onto which a set of timing constrained applications is mapped. On each processor, a scheduling policy is applied, if multiple tasks are mapped onto it.

In order to give a precise problem formulation, we will first present a terminology.

We consider a set of processors interconnected by buses (or other communication channels) the system *architecture* or *(hardware) platform*. Onto this platform, a set of *applications* is to be executed, each consisting of a set of *tasks*, whose relationships are defined by a *task graph*. Furthermore, applications may be temporally constrained. In this case, we speak of *real-time applications*. We consider an architecture together with a set of (real-time) applications a *(real-time) system*. In order to completely describe running real-time systems, a set of design *parameters*, such as task mappings, scheduling parameters (i.e. priorities), or clock rates also need to be defined. We consider a

real-time system together with a complete set of parameters a *system configuration*. A given configuration has a set of *properties*, such as application end-to-end latencies. Note that in each design stage, a different set of system parameters is available to the designer. These will be referred to as *available parameters*. For the sake of simplicity, we will use the term *parameter* equivalent to *available parameter* and account the parameters that are not available in the current design step to the set of *properties*. We consider a given system configuration *feasible*, if all applications adhere to their timing constraints.

The challenge addressed in this paper is to find a methodology for designing adaptive systems that not only ensure functional correctness, but also adhere to system-wide temporal constraints such as end-to-end latencies. With respect to the terminology introduced above, this means finding a methodology that enables a system to verify that its current configuration is feasible, protect itself against transitions into infeasible configurations and ultimately to reconfigure itself to reenter a feasible state. To achieve the latter, the system must perform self-optimization using available parameters during run-time. For our purposes, we assume scheduling parameters, such as priorization or execution sequences to be available as is the case in most real-time kernels. Other parameters, such as task mapping can also be made available by implementing adequate techniques from literature.

From the problem statement, one can deduce the necessary components of such a framework. One needs a feasibility evaluator for a given system configuration, a sensor component, that monitors the current system properties to be fed into the feasibility evaluator, an optimization component in order to generate alternate configurations, as well as an actuator component, that transitions the system from one configuration into another. Furthermore, a framework for the interactions of these components must be put into place.

The feasibility evaluator is the key component in the setup outlined above. It is desirable to use an evaluator, that can not only decide on feasibility, but is also able to compute fitness values for a given system configuration, so that it can also be used by the optimization component.

Furthermore, since we are targeting hard real-time systems and want to give guarantees on real-time performance, the evaluator must use a formal approach to computing the current system properties. As stated in the related work section, current approaches solve this problem in diverse ways. In distributed organic real-time systems, non-centralized solutions to fitness evaluation of a current system configuration that adapt to the system's evolution are preferred over centralized ones, that introduce single points of failure. Thus, only distributable approaches to performance verification are considered for a suitable fitness evaluator.

The next sections will go into detail on the feasibility evaluator and give a closer description of a framework capable of online performance control of an evolving real-time system.

4 Performance Control Framework

We chose the methodology described in [12] as a driving technology for the evaluator for several reasons. The compositional approach is strongly decoupled by efficiently parameterized event models and a distributed analysis algorithm following the approach has already been presented in [13]. Furthermore, the computational load implied by the analysis engine can easily be scaled by applying more or less sophisticated local scheduling analysis techniques. For static priority scheduling this could mean taking inter-event-correlations into account or simply performing a context-blind schedulability analysis. Both approaches yield conservative results for local worst-case response times, but with different accuracy. This opens the possibility to trade analysis accuracy for computational load.

In order to build a system model compliant with the analysis approach, for each task, a worst-case execution time, the activation scheme described by a standard event model ([12]), its communication partners, as well as the maximum communication volume with each partner must be known. The same is true for scheduling policies on each shared resource. We assume that these values are annotated to the task set, although we do not go into detail on how these values are found. Possibilities range from formal analysis [16], to extensive offline simulation and tracing. These methods are already successfully applied for design-time system timing analysis by early adaptors of formal methods e.g. in the automotive industry. In case of real-time constrained applications, we assume that the applicable constraints are also annotated to the task set.

In order to enable adaptation in evolving real time systems, the feasibility evaluator must be embedded in a framework for online real-time control. We divide the structure of the control framework into three major parts, an observer, a controller and an analysis layer (see figure 1). The actual real-time systems is depicted as SuOC - the “System under Observation and Control” [4]. An Observer continuously monitors the systems behaviour to build and maintain an analysable model of the current configuration ("monitor component"). This model is analysed by the formal analysis layer. The results of the analysis are in turn used by a Controller to monitor whether the system complies with all temporal constraints. Thus, the analysis engine, together with part of the Controller form the “feasibility evaluator". For continuous self-optimization, the controller can use the analysis layer to perform optimizations based on the current system model. If optimization results in a new (better) configuration, it is also the Controllers task to inject the new configuration into the system ("actuator component").

Using this framework, one can implement self-awareness and self-protection with respect to timing properties of the current system configuration in an embedded system. Self-awareness is achieved by maintaining a formally analysed model of the system at all times, which can also be used to perform what-if analysis before admitting new applications into the system resulting in self-protecting properties of the embedded system. The next paragraphs elaborate on these concepts.

From the annotated information of each application, partial models corresponding to the task set running on the local processor are generated by local observer instances. As the key metrics needed for building the model are annotated to the task set, the main challenge in generating a complete, distributed model is establishing connectivity between the partial models as well as synthesizing models for the communication

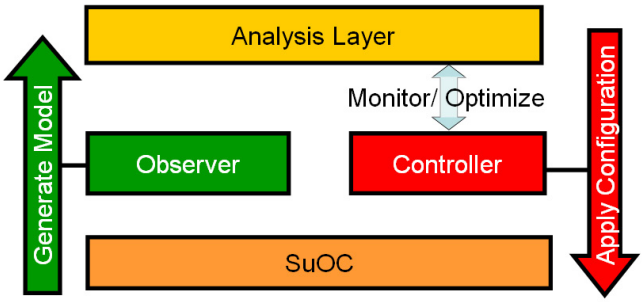


Fig. 1. Framework Architecture

infrastructure from the distributed information about communication partners and volumes.

Before an application is accepted to be mapped on the platform, the current system model is extended by the application and tested for feasibility. If no constraints are violated in the model, the application may execute and is guaranteed to meet its constraints, as long as no application in the system violates its timing properties as annotated. We consider this construct a *service contract* between the system and the applications. This construct ensures that the system will only transition from one provenly save configuration into the next provenly save configuration, thus introducing self-protection into evolving real-time systems.

To ensure compliance with the service contracts, we propose to implement local watchdogs monitoring execution times and communication volumes as well as activation frequencies. The observed values will continuously be compared with the information forming the service contracts of the individual applications. In case a violation of a service contract of an application is detected, a controller is notified, in order to take immediate action. Possibilities range from shaping the load implied by the application to the load defined in the service contract (thus achieving isolation from the other applications), to stopping the application.

At the same time, the current system model is updated to reflect the newly observed configuration. If the resulting system still complies with all given constraints, the application may be readmitted into the system with an adapted service contract. Otherwise, optimization algorithms may be used to find a feasible configuration.

As violations of service contracts may not only be caused by faulty application annotations, but also by component failures or degradation, the above techniques also constitute a self-healing technique efficiently using slack present in the system to cope with component faults and failures. The efficiency of this technique directly scales with the power of the system optimization algorithms put into place.

Figure 2 shows a more detailed view on the architecture of the performance control framework. Distributed observer instances generate partial models of their local environment that are communicated to local analysis engines. These engines, in turn cooperate to perform a distributed system-wide performance analysis of the currently

observed system configuration as described in [13]. Clearly, actuator components also need to be distributed over the whole system, in order to efficiently perform system configuration transitions, thus, the controller must also be implemented distributedly. The cooperating observers, controllers and analysis engines form a global performance control plane.

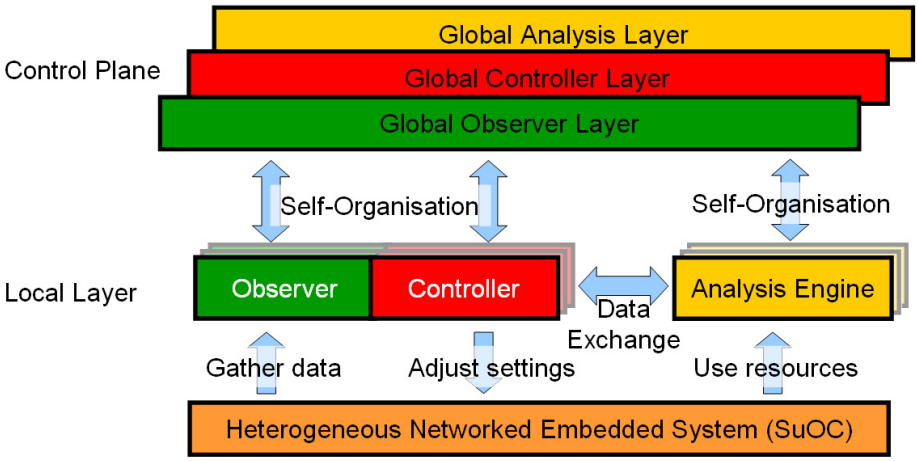


Fig. 2. Control Framework

5 Analysis Methodology

We use the analysis technique proposed by Richter et al [12] to form the global analysis and evaluator plane. A general approach to distributed performance analysis using this technique has been proposed in [13]. As this approach is discussed in the experimental section, we give a short overview on the approach to distributed performance analysis in the next paragraphs. First, the SymTA/S approach is introduced shortly, then the extensions for distributed computation are outlined.

The compositional performance analysis methodology used for this project, solves the global system-level performance verification problem by decomposing the system into independently investigated components.

Each Processor or Bus is modeled as a component (*computation, communication resource*) that may contain tasks. The possible I/O timing between the tasks (*event streams*) is captured with event models that can efficiently be described by a small set of parameters.

Input event models capture event patterns leading to task activations. These are used to perform a local scheduling analysis of a resource to derive the local response times as well as *output event models*.

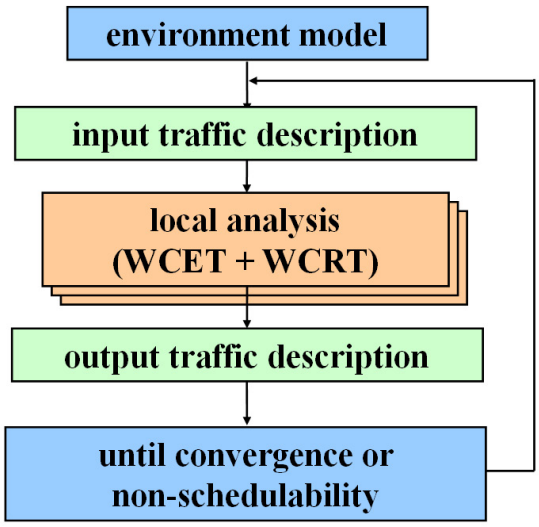


Fig. 3. Analysis Loop

These output event models are propagated to subsequent resources where they are used, in turn, as input event models. In setups with cyclic dependencies the assumed event streams become increasingly more generic. This procedure either converges (and provides a conservative estimation of system properties such as jitter and latencies which can be checked against given constraints), or the system's schedulability can not be guaranteed. Figure 5 shows the structure of the analysis loop as implemented in the tool.

The analysis of a SymTA/S model can easily be distributed over multiple analysis engines, as local scheduling analysis runs are strongly decoupled by event streams. A method to connect partial models managed by multiple analysis engines has been proposed in [13]. Here, it is proposed to tunnel event stream information between multiple analysis engines using existing communication infrastructure. Distributed analysis control performs a local scheduling analysis on a resource as soon as an input event stream changes. As a major advantage, this scheme is naturally adapted to the underlying platform topology and can follow its evolution, as communication with other analysis engines is only necessary, if mapped applications communicate over an existing link. Thus, if communication between analysis engines is necessary, suitable infrastructure must be present.

6 Experimental Study

In this section, we take a closer look at the expected computational load an embedded SymTA/S analysis engine will impose on an embedded system. To do this, we implemented the distributed control algorithm as proposed in [13] by extending the offline

tool. Performing schedulability analysis is the compute intensive part of the iteration loop. Thus, the load imposed by an analysis engine scales with the number of schedulability analysis runs needed to analyse a given system and the load imposed by a single schedulability analysis run. Here, we want to assess the quality of distributed algorithm steering the iteration. As a quality measure for a given analysis control algorithm, we propose the convergence speed of the system wide performance analysis, as measured by the number of schedulability analysis runs needed to analyse the properties of this system. We consider a control algorithm optimal for a given problem, if it solves the global fix-point iteration with a minimal number of schedulability analysis runs. This also implies that an optimal control algorithm imposes the minimal load for a given system and schedulability analysis algorithm implementation.

For testing, we used an in-house system generator tool to generate analysable system models. The generated systems contain a configurable amount of connected tasks an resources. For testing purposes, we generated systems, scaling them in the number of tasks, resources and length of task chains. To benchmark the distributed analysis control mechanism, we analysed these systems using the distributed approach as well as the centralized approach implemented in the tool. We assume that the centralized approach performs close to optimal.

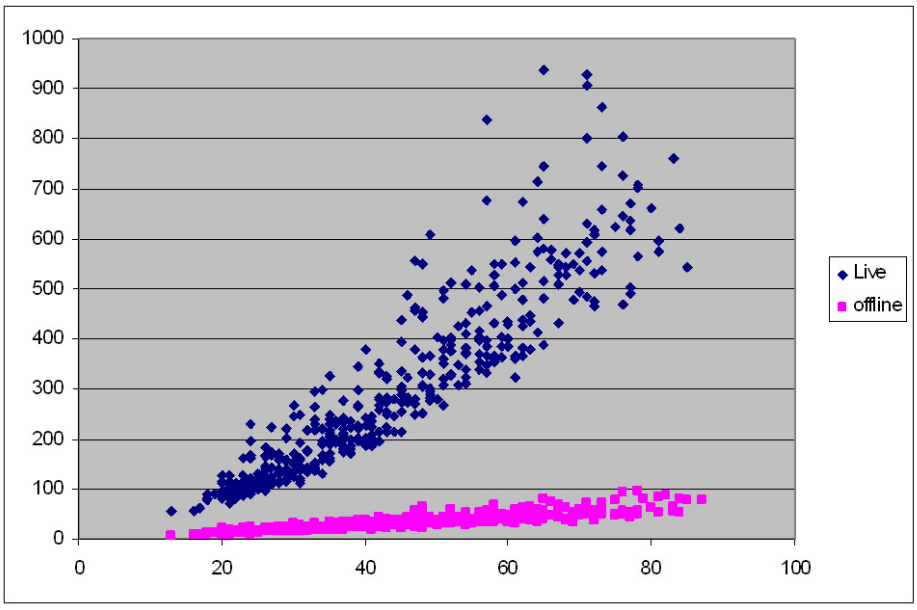
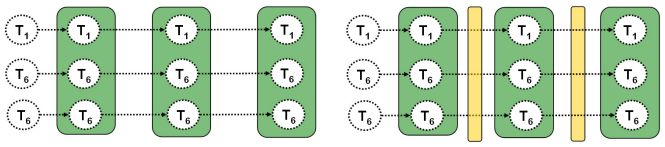


Fig. 4. Performance naive algorithm

The result of a first test can be seen in figure 4. It shows the number of schedulability analysis runs needed for a complete system analysis over increasing system size. The upper point cloud shows the performance of the naive algorithm as proposed in [13], the lower one shows the performance of the offline algorithm as implemented in the tool SymTA/S.



(a) naiv setup (b) insert buffers

Fig. 5. Chained System

The distributed approach shows weak performance w.r.t. schedulability analysis runs needed to analyse big systems. A closer look at possible causes reveals a system configuration that requires an exponentially growing number of schedulability runs to be analysed if using the distributed performance analysis control algorithm, where theoretically a linear relationship suffices: Suppose a system consisting of a series of resources that host a number of independent task chains as depicted in figure 5(a). The system can be scaled in two dimensions - the number of resources and the number of parallel task chains. The minimum number of schedulability analysis runs that is needed scales linearly with the number of resources in the system, as each resource only needs to be analysed once (from left to right). Increasing the number of parallel chains does not have any effect on the number of schedulability analysis runs needed. The proposed distributed algorithm, however shows in part exponential behaviour with increasing number of parallel task chains (see figure 6), since each scheduling analysis on a resource recalculates the output event models of all n tasks on it, and thus potentially triggers renewed analysis on the succeeding resource n times, where one analysis run would be sufficient.

A solution to this problem is to introduce buffers between successing resources as shown in figure 5(b) that collect the changes of incoming event streams resulting from one schedulability analysis on a preceeding resource and release them in as on event reducing the number of reanalysis events for the succeeding resource to one. This approach reduces the number of analysis runs needed to the theoretical minimum for this class of systems.

We implemented the buffering scheme in our offline prototype and redid the experiments outlined above. The results as shown in figure 7 show that this improvement to the distributed analysis control already yields convergence speeds comparable to those of the offline tool for the class of systems produced by our system generator. Interestingly, the distributed control algorithm sometimes even outperforms the centralized algorithm. This is due to the fact that the centralized algorithm does not exploit all knowledge about the system model to precompute an optimal sequence of schedula-

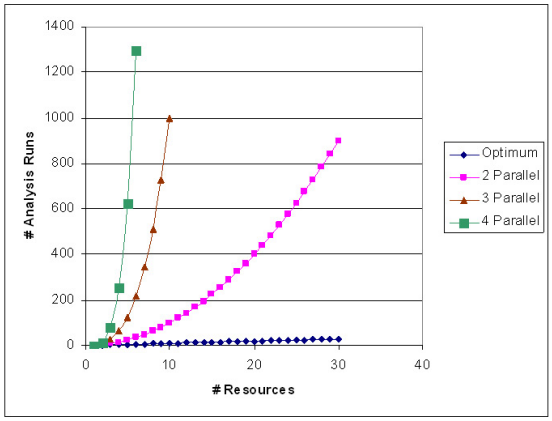


Fig. 6. Performance in number of analysis runs

bility analysis runs. This would imply computing a topological sort on the resource graph. Thus the distributed algorithm can outperform the centralized one, if the activation scheme coincidentally follows a topological sort of the graph. The fact that the experiments contained many of these cases may be due to the regular structure of the generated system models.

Further improvements to both, the offline and the distributed control algorithms can be made by first performing a topological sort of the resources in the system model, that can be used to determine an order of local scheduling analysis runs, yielding an optimal control algorithm.

7 Conclusion

In this paper, we introduced a framework that enables the implementation of organic real-time systems. It is sensitive to changes in the hardware architecture as well as software configuration of the system.

It is based on a layered architecture of observers and controllers and a distributed analysis layer that evaluates the local analysis results and event model parameters. Experiments with a prototype implementation of the analysis methodology to be used have shown, that the computational load implied remains small.

The presented approach is suitable for implementing evolving hard real-time that are capable of self-protection against transitions into non-feasible system states w.r.t. timing properties.

References

1. Larry Bull and Tim Kovacs. *Foundations of Learning Classifier Systems*. Springer, Berlin, July 2005.

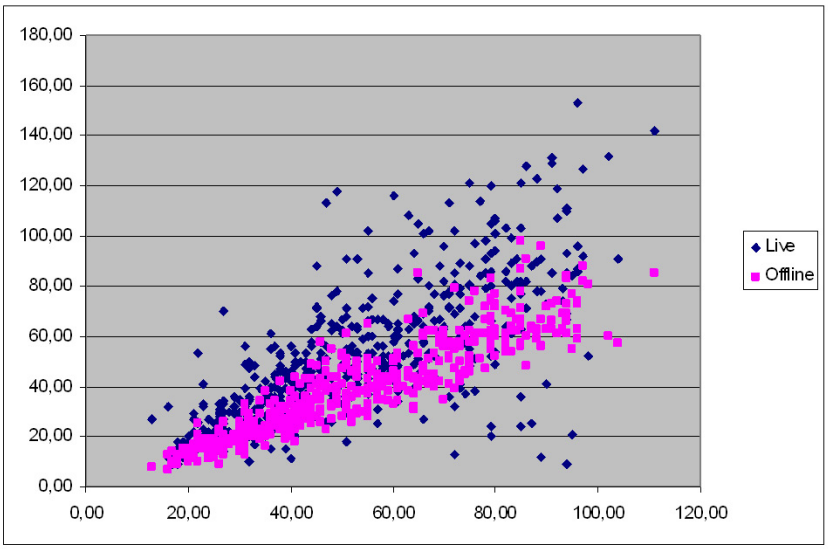


Fig. 7. Performance with buffering

2. J. Ellebæk, K. S. Knudsen, A. Brekling, M. R. Hansen, and J. Madsen. MOVES - a tool for modeling and verification of embedded systems. In *DATE'07 University Booth*, apr 2007.
3. John A. Stankovic et al. Feedback control scheduling in distributed real-time systems. In *RTSS '01: Proceedings of the 22nd IEEE Real-Time Systems Symposium (RTSS'01)*, page 59, Washington, DC, USA, 2001. IEEE Computer Society.
4. Jürgen Branke et al. Organic computing - addressing complexity by controlled self-organization. In Tiziana Margaria, Anna Philippou, , and Bernhard Steffen, editors, *Proceedings of ISoLA 2006*, IEEE-ISoLA, pages 200–206, Paphos, Cyprus, NOV 2006.
5. W. Stechele et al. Concepts for autonomic integrated systems. In *edaWorkshop*, Hannover, Germany, 19 - 20 June 2007.
6. Arne Hamann, Razvan Racu, and Rolf Ernst. A formal approach to robustness maximization of complex heterogeneous embedded systems. In *International Conference on Hardware/Software Codesing and System Synthesis (CODES+ISSS)*, oct 2006.
7. C. Haubelt, D. Koch, and J. Teich. Reconet: modeling and implementation of fault tolerant distributed reconfigurable hardware. In *16th Symposium on Integrated Circuits and Systems Design*, 2003.
8. Paul Horn. Autonomic computing: Ibm's perspective on the state of information technology. *http://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf*, October 2001.
9. Kim G. Larsen, Paul Pettersson, and Wang Yi. Uppaal in a nutshell. *International Journal on Software Tools for Technology Transfer (STTT)*, 1:134–152, 1997.
10. Christian Mueller-Schloer. Organic computing - on the feasibility of controlled emergence. In *IEEE/ACM/IFIP International Conference on Hardware/Software Codesing and System Synthesis (CODES + ISSS 2004)*, 2004.
11. P. Poplavko, T. Basten, M. Bekooij, J. van Meerbergen, and B. Mesman. Task-level timing models for guaranteed performance in multiprocessor networks-on-chip. In *CASES '03:*

Proceedings of the 2003 international conference on Compilers, architecture and synthesis for embedded systems, pages 63–72, New York, NY, USA, 2003. ACM Press.
12. Kai Richter. *Compositional Scheduling Analysis Using Standard Event Models*. PhD thesis, Technical University of Braunschweig, Department of Electrical Engineering and Information Technology, 2004.
13. Steffen Stein, Arne Hamann, and Rolf Ernst. Real-time property verification in organic computing systems. In *2nd IEEE International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA)*, November 2006.
14. David Tennenhouse. Proactive computing. *Commun. ACM*, 43(5):43–50, 2000.
15. L. Thiele, S. Chakraborty, and M. Naedele. Real-time calculus for scheduling hard real-time systems. In *International Symposiumon Circuits and Systems (ISCAS)*, 2000.
16. R. Wilhelm et al. The worst-case execution time problem — overview of methods and survey of tools. Technical report, Malardalen Real-Time Research Centre, Malardalen University, March 2007.