Parametric Schedulability Analysis of Fixed Priority Real-Time Distributed Systems*

Youcheng Sun[1], Romain Soulat[2], Giuseppe Lipari[1,2], Étienne André[3], and Laurent Fribourg[2]

[1] Scuola Superiore Sant'Anna, Pisa, Italy
[2] LSV, ENS Cachan & CNRS, France
[3] Université Paris 13, Sorbonne Paris Cité, LIPN, CNRS, Villetaneuse, France

Abstract. In this paper, we address the problem of parametric schedulability analysis of distributed real-time systems scheduled by fixed priority. We propose two different approaches to parametric analysis. The first one is a novel analytic technique that extends single-processor sensitivity analysis to the case of distributed systems. The second approach is based on model checking of Parametric Stopwatch Automata (PSA): we generate a PSA model from a high-level description of the system, and then we apply the Inverse Method to obtain all possible behaviours of the system. Both techniques have been implemented in two software tools, and they have been compared with classical holistic analysis on two meaningful test cases. The results show that the analytic method provides results similar to classical holistic analysis in a very efficient way, whereas the PSA approach is slower but covers the entire space of solutions.

1 Introduction and Motivation

Designing and analysing distributed real-time systems is a very challenging task. The main source of complexity arises from the large number of parameters to consider: tasks priorities, computation times and deadlines, synchronisation, precedence and communication constraints, etc. Finding the optimal values for the parameters is not easy and often a small change in one parameter may completely change the behaviour of the system and even compromise its correctness. For these reasons, designers are looking for analysis methodologies that allow incremental design and exploration of the parameter space.

Task computation times are particularly important parameters. In modern processor architectures, it is very difficult to precisely compute worst-case computation times of tasks, thus estimations derived by previous executions are often used in the analysis. However, estimations may turn out to be optimistic, hence an error in the estimation of a worst-case execution time may compromise the schedulability of the entire system.

* This is the author version of the paper of the same name accepted for publication at FTSCS 2013. The final publication is available at http://www.springer.com.

In this paper we investigate the problem of doing parametric analysis of real-time distributed systems scheduled by fixed priority. We consider an application modelled by a set of pipelines (also called *transactions* in [PGH98]), where each pipeline is a sequence of periodic tasks to be executed in order, and all tasks in a pipeline must complete before an end-to-end deadline. We consider that all processors in the distributed system are connected by one or more CAN bus [DBBL07], a network standard used in automotive applications.

The first contribution of the paper (Section 4) is to propose a new method for doing parametric analysis of the system, using the worst-case computation times of the tasks as parameters. The method extends the sensitivity analysis proposed by Bini *et al.* [Bin04] by considering distributed systems and non-preemptive scheduling.

The proposed analytical method is not exact, as it sometimes overestimates the interference of higher priority tasks and of previous tasks in the pipeline on the response time of a task. Therefore, the second contribution of the paper (Section 5) is to propose also an exact schedulability analysis by modelling a distributed real-time system as a set of parametric timed automata; then we apply a model checking methodology using the Inverse Method [AS13,FLMS12].

Finally, in Section 6 we compare these two approaches with the MAST tool [GHGPD01,Gru], a state-of the art tool for classical schedulability analysis. Comparison is performed on two case studies from the research literature on which we measured run-time and effectiveness of the three analyses. Results show that the analytical approach can very efficiently compute the feasible space of parameters with a good precision.

2 Related Work

Many research papers have already addressed the problem of parametric schedulability analysis, especially on single processor systems. Bini and Buttazzo [Bin04] proposed an analysis of fixed priority single processor systems, which is used as a basis for this paper.

Parameter sensitivity can be also be carried out by repeatedly applying classical schedulability tests, like the holistic analysis [PGH98]. One example of this approach is used in the MAST tool [GHGPD01], in which it is possible to compute the *slack* (i.e. the percentage of variation) with respect to one parameter for single processor and for distributed systems by applying binary search in that parameter space [PGH98].

A similar approach is followed by the SymTA/S tool [HHJ+05], which is based on the *event-stream* model [RE02]. Another interesting approach is the Modular Performance Analysis (MPA) [WTVL06], which is based on Real-Time Calculus. In both cases, the analysis is compositional, therefore less complex than the holistic analysis. Nevertheless, these approaches are not fully parametric, in the sense that it is necessary to repeat the analysis for every combination of parameter values in order to obtain the schedulability region.

Model checking of *parametric timed automata* (PTA) or *parametric stopwatch automata* (PSA) can be used for parametric schedulability analysis [CPR08]. In particular, thanks to generality of the PTA and PSA modelling language, it is possible to model a larger class of constraints, and perform parametric analysis on many different variables, for example task offsets. This approach has been recently extended to distributed real-time systems [LPPR13].

Also grounded on PTA and PSA is the Inverse Method [AS13], applied in particular to schedulability analysis [FLMS12]. This method is very general because it permits to perform analysis on any system parameter. However, this generality may be paid in terms of complexity of the analysis.

In this paper, we aim at performing fully parametric analysis of real-time distributed systems. We first present extensions of the methods proposed in [Bin04] to the case of distributed real-time systems. We also present a model of a distributed real-time systems using PSA, and compare the two approaches against classical analysis in MAST.

3 System Model

We consider distributed real-time systems consisting of several computational nodes, each one hosting one single processor, which are connected by one or more shared networks. Without loss of generality, from now on we will use the term *task* to denote both tasks and messages, and the term *processor* to denote both processors and networks.

A distributed real-time system consists of a set of task pipelines $\{\mathcal{P}^1, \ldots, \mathcal{P}^n\}$ to be executed on a set of processors. A *pipeline* is a chain of tasks $\mathcal{P}^j = \{\tau_1^j, \ldots, \tau_n^j\}$ to be executed in order, and each task is allocated on one (possibly different) processor. In order to simplify the notation, in the following we sometimes drop the pipeline superscript when there is no possibility of misinterpretation.

A pipeline is assigned two fixed parameters: T^j is the pipeline period and D^j_{e2e} is the end-to-end deadline. This means that all tasks of the pipeline are activated together every T^j units of time; and all tasks should be completed within a time interval of D^j_{e2e}.

A task in the pipeline can be a piece of code to be executed on a processor or a message to be sent over a network. More precisely, a real-time periodic task is a tuple $\tau_i = (C_i, T_i, D_i, R_i, q_i, p_i, J_i)$.

This task model contains the following fixed parameters:

- T_i is the task period. All tasks in the same pipeline have period equal to the pipeline period T;
- D_i is the task relative deadline;
- q_i is the task priority; the larger q_i, the higher the priority;
- p_i is the index of the processor (or network) on which the task executes.

Also, a task is characterised by the following free parameters (variables):

- C_i is the worst-case computation time (or worst-case transmission time, in case it models a message). It is the worst-case time the task needs to complete one periodic instance when executed alone on a dedicated processor (or network). In this paper we want to characterise the schedulability of a distributed system in the space of the computation times, so C_i is a free parameter.
- R_i is the *task worst-case response time*, i.e. the worst case finishing time of any task instance relative to the activation of its pipeline.
- J_i is the task worst-case activation jitter, i.e. the greatest time since its activation that a task must wait for all preceding tasks to complete their execution.

Every task activation is an *instance* (or *job*) of the task. We denote the kth instance of task τ_i as $\tau_{i,k}$. An instance $\tau_{i,k}$ of a task in the pipeline can start executing only after the corresponding instance of the preceding task $\tau_{i-1,k}$ has completed. Finally, the last task in the pipeline must complete every instance before D_{e2e} units of time from its pipeline's activation. For a job $\tau_{i,k}$ we define the following notation:

- $a_{i,k}$ is $\tau_{i,k}$'s arrival time (coincident with the activation time of the pipeline).
- $s_{i,k}$ is the start time of the instance, i.e. the first time the instance executes on the processor.
- $f_{i,k}$ is the job's finishing time.
- $r_{i,k}$ the task release time. The first task of a pipeline is released immediately at the time of its arrival $r_{0,k} = a_{0,k}$; successive tasks are released at the finishing time of the preceding tasks: $r_{i,k} = f_{i-1,k}$. The following relationship holds: $\forall i, k \;\; a_{0,k} = a_{i,k} \leq r_{i,k} \leq s_{i,k} < f_{i,k}$
- The maximum difference between arrival and release time is the worst-case activation jitter of the task: $J_i = \max_k(r_{i,k} - a_{i,k})$.
- The maximum difference between finishing time and arrival time is the worst-case response time of the task: $R_i = \max_k(f_{i,k} - a_{i,k})$.

Parameters R_i and J_i depend on the other tasks parameters and on the scheduling policy according to a complex set of equations. Of course, they cannot be considered parameters that the programmer can modify: nevertheless, for our purposes it is useful to consider them as variables to help us write the set of constraints that define the schedulability space (the exact role of such variables will be detailed in Section 4.3).

A scheduling algorithm is *fully preemptive* if the execution of a lower priority job can be suspended at any instant by the release of a higher priority job, which is then executed in its place. A scheduling algorithm is *non-preemptive* if a lower priority job, once it has started executing, can complete its execution regardless of the release of higher priority jobs. In this paper, we consider fully preemptive fixed priority scheduling for processors, and non-preemptive fixed priority scheduling for networks.

4 Analytic Method

In this section we present a novel method for parametric analysis of distributed system. The method extends the sensitivity analysis by Bini *et al.* [SLS98,Bin04] to include jitter and deadline parameters.

In Sections 4.1 and 4.2, we only consider the scheduling of independent periodic tasks in a single processor. Then, in Section 4.3, we extend the schedulability analysis to distributed systems.

4.1 Preemptive Tasks with Constrained Deadlines

There are many ways to test the schedulability of a set of real-time periodic tasks scheduled by fixed priority on a single processor. In the following, we will use the test proposed by Seto *et al.* [SLS98] because it is amenable to parametric analysis of computation times, jitters and deadlines.

The original theorem was formulated for tasks with deadlines equal to periods. For the moment, we generalise it to tasks with constrained deadlines (i.e. $D_i \leq T$), while in Section 4.2 we deal with unconstrained deadlines, jitter and non-preemptive scheduling.

Definition 1. *The set of* scheduling points $\mathbb{P}^{i-1}(t)$ *for a task* τ_i *is the set of all vectors corresponding to multiples of the period of any task* τ_j *with priority higher than* τ_i*, until the maximum possible value of the deadline. It can be computed as follows. Let* $\eta_j(t) = \left\lceil \frac{t}{T_j} \right\rceil$*, and let* $\eta^{i-1}(t)$ *be the corresponding vector of* $i-1$ *elements with* $j = 0, \ldots, i-1$*. Then:*

$$\mathbb{P}^{i-1}(t) = \{\eta^{i-1}(t)\} \cup \{\eta^{i-1}(kT_h) \mid 0 < kT_h < t, h < i\} \tag{1}$$

Theorem 1 ([SLS98]). *Consider a system of periodic tasks* $\{\tau_1, \ldots, \tau_n\}$ *with constrained deadlines and zero jitter, executed on a single processor by a fixed priority preemptive scheduler. Assume all tasks are ordered in decreasing order of priorities, with* τ_1 *being the highest priority task.*

Task τ_i *is schedulable if and only if:*

$$\exists \mathbf{n} \in \mathbb{P}^{i-1}(D_i) \begin{cases} C_i + \sum_{j=1}^{i-1} n_j C_j \leq n_k T_k \quad \forall k = 1, \ldots, i-1 \\ C_i + \sum_{j=1}^{i-1} n_j C_j \leq R_i \\ R_i \leq D_i \end{cases} \tag{2}$$

where $\mathbf{n}$ *is a vector of* $i-1$ *integers, and* $\mathbb{P}^{i-1}(D_i)$ *is the set of scheduling points.*

Notice that, with respect to the original formulation, we have separated the case of $k = i$ from the rest of the inequalities and we introduced variable R_i.

The theorem allows us to only consider sets of linear inequalities, because the non-linearity has been encoded in the variables n_j. Each vector $\mathbf{n}$ defines a convex region (maybe empty) with variables $C_1, \ldots, C_i$ and $R_1, \ldots, R_i$. The "*exists*" quantifier means that the region for each task τ_i is the union of convex regions, hence it may be non-convex. Since we have to check the schedulability of all tasks, we must intersect all such regions to obtain the final region of schedulable parameters. The resulting system is a disjunction of sets of conjunctions of inequalities. Geometrically, this corresponds to a non-convex polyhedron in the space of the variables C and R of tasks.

It is worth to note that, using this formulation, we can compute the response time of a task by simply minimising the corresponding variable R_i under the constraints of Equation (2). As an example, consider the following task set (the same as in [BB04]): $\tau_1 = (C = 1, T = 3), \tau_2 = (C = 2, T = 8), \tau_3 = (C = 4, T = 20)$, in decreasing order of priority, to be scheduled by preemptive fixed priority scheduling on a single processor.

We consider the response time R_3 as a parameter and set up the system of inequalities according to Equation (2). After reduction of the non-useful constraints, we obtain $12 \leq R_3 \leq 20$. Therefore, the response time is $R_3 = 12$, which is the same that can be obtained by classical response time analysis.

4.2 Extensions to the Model

We now extend Seto's test to unconstrained deadlines and variable jitters, and non-preemptive scheduling. Non-preemptive scheduling can be modelled by considering an initial *blocking time*, due to the fact that a task cannot preempt lower-priority executing tasks.

The worst case response time for a non preemptive task τ_i can be found in its longest i-level active period [BLV07]. An i-level active period L_i is an interval $[a, b)$ such that the amount of processing that needs to be performed due to jobs with priority higher than or equal to τ_i (including τ_i itself) is larger than 0 for all $t \in (a, b)$, and equal to 0 at instants a and b. The longest L_i can be found by computing the lowest fixed point of a recursive function. Notice that, by considering non-preemption and tasks with deadline greater than periods, the worst-case response time may be found in any instance of the active period, not necessarily in the first one (as with the classical model of constrained deadline preemptive tasks).

Unfortunately, the longest busy period cannot be computed when tasks have parametric worst-case computation times. However, under the assumption that there is at least an idle-time in the hyperperiod (i.e. its utilisation is strictly less than 100%) a sufficient feasibility test can be derived by computing the worst-case response time for every instance of the task set in the hyperperiod H_n. Therefore, we can extend our model as follows.

Theorem 2. *A non preemptive task τ_i is schedulable if $\forall h = 1, \ldots, \frac{H_n}{T_i}$, $\exists \mathbf{n} \in \mathbb{P}^{i-1}((h-1)T_i + D_i)$ such that*

- $B_i + (h-1)C_i + \sum_{j=1}^{i-1} n_j C_j \leq n_l T_l - J_l \quad \forall l = 1, \ldots, i-1$;
- $B_i + (h-1)C_i + \sum_{j=1}^{i-1} n_j C_j \leq (h-1)T_i + R_i - C_i - J_i$;
- $R_i \leq D_i$ *and* $B_i \leq C_j - 1$ *for all* $j > i$.

Proof. See [SSL+13].

Term B_i is an additional internal variable used to model the blocking time that a task suffers from lower priority tasks. It is possible to avoid the introduction of this additional variable by substituting it in the inequalities with a simple Fourier-Motzkin elimination.

Notice that the introduction of unconstrained deadlines adds a great amount of complexity to the problem. In particular, the number of non-convex regions to intersect is now $\mathcal{O}(\sum_{i=1}^{n} \frac{H_n}{T_i})$, which is dominated by $\mathcal{O}(nH_n)$. So, the proposed problem representation is pseudo-polynomial in the size of the hyperperiod. However, in real applications, we expect the periods to have "nice" relationships: for example, in many cases engineers choose periods that are multiples of each others. Therefore, we expect the set of inequalities to have manageable size for realistic problems.

4.3 Distributed Systems

Until now, we have considered the parametric analysis of independent tasks on single processor systems, with computation times, response times, blocking times and jitters as free variables.

One key observation is that a precedence constraint between two consecutive tasks τ_i and τ_{i+1} in the same pipeline can be expressed as $R_i \leq J_{i+1}$. This relationship derives directly from the definition of response time and jitter in Section 3. Using this elementary property, we can now build the parametric space for a distributed system as follows.

1. For each processor and network, we build the constraint system of Theorem 2. Notice that the set of constraints for the individual single processor systems are independent of each other (because they are constraints on different tasks).
2. For each pipeline $\mathcal{P}^a$:
   - two successive tasks τ_i^a and τ_{i+1}^a must fulfil the constraint $R_i^a \leq J_{i+1}^a$;
   - for the initial task we impose $J_1^a = 0$.

Such pipeline constraints must intersect the combined system to produce the final system of constraints. However, simply adding the above precedence constraints can lead to pessimistic solutions. In fact, if two tasks from the same pipeline are assigned to the same processor, the interference they may cause on each other and on the other tasks may be limited.

Suppose τ_i^a and τ_j^a are allocated to the same processor and $q_i^a > q_j^a$. Then, τ_i^a can at most interfere with the execution of a job from τ_j^a a number of times equal to $\xi = \left\lceil \frac{\max\{0, D_{e2e}^a - T^a\}}{T^a} \right\rceil$. So, we impose that $\forall \mathbf{n} \in \mathbb{P}^{j-1},\ n_i \leq \xi$.

The analytic method proposed in this section has been implemented in a software tool, called RTSCAN, which is based on the PPL (Parma Polyhedra Library) [BHZ08], a library specifically designed and optimised to represent and operate on polyhedra. The library efficiently operates on rational numbers with arbitrary precision: therefore, in this work we make the assumption that all variables (computations times, response times and jitter) are defined in the domain of rationals (rather than reals).

We observed that the complexity of the methodology for generating the parameter space strongly depends on the number of free parameters considered in the analysis. Therefore, as a preliminary step, the tool requires the user to select a subset of the computation times on which the analysis will be performed, whereas the other parameters will be assigned fixed values. During construction of the polyhedron we have to keep R_i, J_i and B_i for each task as variables. Therefore, the number of variables to be managed is $\mathrm{nV} = 4 \cdot N + F$, where N is the number of tasks and F is the number of variables to analyse. At the end, we can eliminate the R_i, J_i and B_i variables, hence the final space consists of F dimensions. An evaluation of this tool and of the run-time complexity of the analysis will be presented in Section 6.

The analytic method described so far is not exact. In fact, when dealing with pipelines in a distributed system we may sometimes overestimate the interference of higher priority-tasks on lower priority ones. For this reason, we now present an exact parametric analysis based on PSA and model checking.

5 The Inverse Method Approach

5.1 Parametric Timed Automata with Stopwatches

Timed automata are finite-state automata augmented with clocks, i.e., real-valued variables increasing uniformly, that are compared within guards and invariants with timing delays [AD94]. Parametric timed automata (PTA) [AHV93] extend timed automata with parameters, i.e., unknown constants, that can be used in guards and invariants. We will use here an extension of PTA with *stopwatches* [AM02], where clocks can be stopped in some control states of the automaton.

Given a set X of clocks and a set U of parameters, a constraint C over X and U is a conjunction of linear inequalities on X and U[1]. Given a parameter valuation (or point) π, we write $\pi \models C$ when the constraint where all parameters within C have been replaced by their value as in π is satisfied by a non-empty set of clock valuations.

[1] Note that this is a more general form than the strict original definition of PTA [AHV93]; since most problems for PTA are undecidable anyway, this has no practical incidence, and increases the expressiveness of the formalism.

Definition 2. *A parametric timed automaton with stopwatches (PSA)* $\mathcal{A}$ *is* $(\Sigma, Q, q_0, X, U, K, I, slope, \rightarrow)$ *with* Σ *a finite set of actions,* Q *a finite set of locations,* $q_0 \in Q$ *the initial location,* X *a set of* h *clocks,* U *a set of parameters,* K *a constraint over* U*,* I *the invariant assigning to every* $q \in Q$ *a constraint over* X *and* U*,* $slope : Q \rightarrow \{0,1\}^h$ *assigns a constant slope to every location, and* $\rightarrow$ *a step relation consisting of elements* (q, g, a, ρ, q')*, where* $q, q' \in Q$*,* $a \in \Sigma$*,* $\rho \subseteq X$ *is the set of clocks to be reset, and the guard* g *is a constraint over* X *and* U*.*

The semantics of a PSA $\mathcal{A}$ is defined in terms of states, i.e., pairs (q, C) where $q \in Q$ and C is a constraint over X and U. Given a point π, we say that a state (q, C) is π-compatible if $\pi \models C$. Runs are alternating sequences of states and actions, and traces are time-abstract runs, i.e., alternating sequences of *locations* and actions. The trace set of $\mathcal{A}$ corresponds to the traces associated with all the runs of $\mathcal{A}$. Given $\mathcal{A}$ and π, we denote by $\mathcal{A}[\pi]$ the (non-parametric) timed stopwatch automaton where each occurrence of a parameter has been replaced by its constant value as in π. Details can be found in, e.g., [AS13].

The Inverse Method for PSA [AS13] exploits the knowledge of a reference point of timing values for which the good behaviour of the system is known. The method synthesises automatically a dense space of points around the reference point, for which the discrete behaviour of the system, that is the set of all the admissible sequences of interleaving events, is guaranteed to be the same.

The Inverse Method *IM* proceeds by exploring iteratively longer runs from the initial state. When a π-incompatible state is met (that is a state (q, C) such that $\pi \not\models C$), a π-incompatible inequality J is selected within the projection of C onto U. This inequality is then negated, and the analysis restarts with a model further constrained by $\neg J$. When a fixpoint is reached, that is when no π-incompatible state is found and all states have their successors within the set of reachable states, the intersection of all the constraints onto the parameters is returned. The algorithm is recalled in Appendix A.

Although the principle of *IM* shares similarities with sensitivity analysis, *IM* proceeds by iterative state space exploration. Furthermore, its result comes under the form of a fully parametric constraint, in contrast to sensitivity analysis. By repeatedly applying the method, we are able to decompose the parameter space into a covering set of "tiles", which ensure a uniform behaviour of the system: it is sufficient to test only one point of the tile in order to know whether or not the system behaves correctly on the whole tile. This is known as the *behavioural cartography* [AF10].

5.2 Modelling the System Using Parametric Stopwatch Automata

Timed Automata with Stopwatches have been used for modelling scheduling problems in the past. Our model technique is similar to [AM02,AAM06], except that we model pipelines of tasks, and that we use PSA for obtaining the space of feasible computation times. In the current implementation, we only model pipelines with end-to-end deadlines no larger than their periods. This allows us

to simplify the model and reduce the complexity of the analysis. The extension to deadlines larger than the period is discussed at the end of the section.

We illustrate our model with the help of an example of two pipelines $\mathcal{P}^1, \mathcal{P}^2$ with $\mathcal{P}^1 = \{\tau_1, \tau_2\}$, $\mathcal{P}^2 = \{\tau_3, \tau_4\}$, $p(\tau_1) = p(\tau_4) = p_1$, $p(\tau_2) = p(\tau_3) = p_2$, p_1 being a preemptive processor and p_2 being non-preemptive. We have that $q_1 > q_4$ and $q_3 > q_2$.

Figure 1 shows the PSA model of a pipeline. A pipeline is a sequence of tasks that are to be executed in order: when a task completes its instance, it instantly releases the next task in the pipeline. Since we assume constrained deadlines, once every task in the pipeline has completed, the pipeline waits for the next period to start. This PSA contains one local clock $x_{\mathcal{P}^1}$, one parameter T_1 (the pipeline's period), and synchronises on 5 actions: "τ_1 release", "τ_1 completed", "τ_2 release", "τ_2 completed", and "$\mathcal{P}^1$ restart". The order of these events imposes that task τ_1 must be entirely executed before task τ_2. The initialisation of the pipeline's local clock $x_{\mathcal{P}^1}$ and the invariant $x_{\mathcal{P}^1} \leq T_1$ ensure that the pipeline's execution terminates within its period T_1. The guard $x_{\mathcal{P}^1} == T_1$ ensures that the pipeline restarts after exactly T_1 units of time.

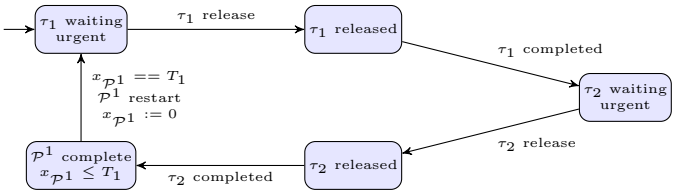

Fig. 1. PSA modelling a pipeline $\mathcal{P}^1$ with two tasks τ_1, τ_2

Figure 2 shows the model of a preemptive processor with 2 tasks τ_1 and τ_4, where task τ_1 has higher priority over task τ_4. The processor starts by being *idle*, waiting for a task release. As soon as a request has been received (e.g. action "τ_4 release"), it moves to one of the states where the corresponding task is running ("τ_4 running"). If it receives another release request ("τ_1 release"), it moves to the state corresponding to the higher priority task running ("τ_1 release, τ_4 released"). The fact that τ_1 does not execute anymore is modelled by the blocking of the clock x_{τ_4} corresponding to task τ_4. Moreover, while a task executes, the scheduler automaton checks if the corresponding pipeline misses its deadline (e.g. guard $x_{\mathcal{P}^1} > D^1_{e2e}$, where D^1_{e2e} is τ_1's deadline). In the case of a deadline miss, the processor moves to a special failure state ("deadline missed") and stops any further computation.

The model of a non-preemptive processor is very similar to the model of preemptive processor: the central state in Figure 2 which accounts for the fact that τ_4 is stopped when τ_1 is released, in the non-preemptive case must not stop τ_4, but simply remember that τ_1 has been released, so that we can move to the top state when τ_4 completes its instance.

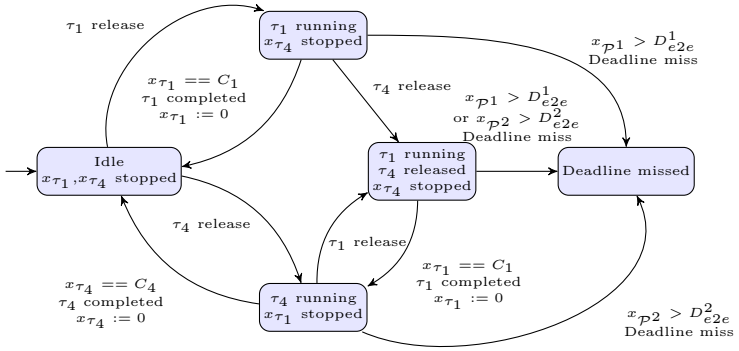

Fig. 2. PSA modelling a preemptive processor with two tasks τ_1, τ_4

We use the IMITATOR software tool [AFKS12] implementing the behavioural cartography, to perform the analysis of the PSA. The tool takes as input a textual description of the PSA and an interval of values for each parameter, which can be seen as a hypercube in $|U|$ dimensions, with $|U|$ the number of parameters. Then, it explores the hypercube of values using IM, and it outputs a set of tiles.

For each tile, IMITATOR derives whether the corresponding system behaviour is valid (i.e. no deadline miss is present), which corresponds to a good tile, or invalid (at least one deadline miss has been found), which corresponds to a bad tile. Every behaviour can be regarded as a set of traces of the system. Although deadline misses are *timed* behaviours, they are reduced to (untimed) traces thanks to the "deadline miss" location of the processor PSA. All points inside one particular tile are values of the parameters that generate equivalent behaviours (they correspond to the same trace set).

The result of the behavioural cartography is a set of tiles that covers "almost"[2] the entire hypercube. The region of space we are looking for is the union of all the good tiles.

The proposed model can be extended to deal with deadlines greater than periods by changing the automaton in Figure 1. In particular, we must take into account that each task can have up to $\lceil \frac{D_{e2e}}{T} \rceil$ pending instances that have not completed yet. However, the number of locations increases with $\lceil \frac{D_{e2e}}{T} \rceil$ and thus the complexity of the analysis.

[2] Technically, a part might be non-covered in some cases at the border between the good and the bad subspace; this part has a width of at most ϵ, where ϵ is an input of the tool; of course, the smaller ϵ, the more costly the analysis (see [AF10,AS13]).

Table 1. Test case 1; all numbers in "ticks"

Pipeline/Task	T	D_{e2e}	Tasks	C	q	p
τ_1	20	20	–	free	9	1
P^1	150	150	τ_1^1	free	3	1
			τ_2^1	10	9	2
			τ_3^1	8	5	3
			τ_4^1	15	2	2
			τ_5^1	25	2	1
τ_2	30	30	–	6	9	3
τ_3	200	200	–	40	2	3

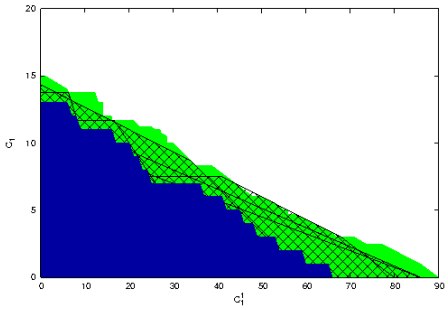

Fig. 3. TC1: Schedulability regions produced by RTSCAN (hatched), MAST (dark blue, below), and IMITATOR (light green, above)

6 Evaluation

In this section we evaluate the effectiveness and the running time of the two proposed tools on two case studies. As a baseline comparison, we choose to also run the same kind of analysis on the same case studies using MAST.

In order to simplify the visualisation of the results, for each test case we present the 2D region generated for two parameters only. However, all three methods are general and can be applied to any number of parameters. In Section 6.3 we will present the execution times of the three tools on the test-cases.

MAST [GHGGPGDM01] is a software tool implemented and maintained by the CTR group at the *Universidad de Cantabria* that allows to perform schedulability analysis for distributed real-time systems. It provides the user with several different kinds of analysis. For our purposes, we have selected the "Offset Based analysis" [PGH98], an improvement over classical holistic analysis that takes into account some of the relationships between tasks belonging to the same pipeline.

6.1 Test Case 1

The first test case (TC1) has been adapted from [PGH98] (we reduced the computation times of some tasks to position the system in a more interesting schedulability region). It consists of three simple periodic tasks and one pipeline, running on two processors (p_1 and p_3), connected by a CAN bus (p_2). The parameters are listed in Table 1. The pipeline models a remote procedure call from processor 1 to processor 3. All tasks have deadlines equal to periods, and also the pipeline has end-to-end deadline equal to its period. Only two messages are sent on the network, and according to our optimisation rule for building parametric space, if the pipeline is schedulable, they cannot interfere with each other. We performed parametric schedulability analysis with respect to C_1 and C_1^1.

The resulting regions of schedulability from the three tools are reported in Figure 3. In this particular test, RTSCAN dominates MAST. After some debug-

Table 2. Test case 2: periods and deadlines are in milliseconds, computation times in micro-seconds.

Pipeline	T	D_{e2e}	Tasks	C	q	p
P^1	200 (30)	200	τ_1^1	4,546	10	1
			τ_2^1	445	10	2
			τ_3^1	9,091	10	4
			τ_4^1	445	9	2
			τ_5^1	free	9	1
P^2	3,000	1,000	τ_1^2	free	9	4
			τ_2^2	889	8	2
			τ_3^2	44,248	10	3
			τ_4^2	889	7	2
			τ_5^2	22,728	8	1

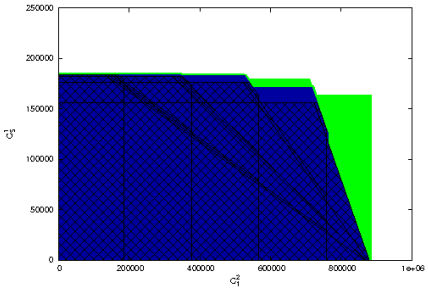

Fig. 4. Schedulability regions for test case 2a, produced by RTSCAN (hatched), MAST (dark blue, below), and IMITATOR (light green, above)

ging, we discovered that the analysis algorithm currently implemented in MAST does not consider the fact that the two messages τ_2^1 and τ_4^1 cannot interfere with each other, and instead considers a non-null blocking time on the network.

As expected, the region computed by IMITATOR dominates the other two tools. This means that there is much space for improvement in the analysis even for such simple systems.[3]

6.2 Test Case 2

The second test case is taken from [WTVL06]. It consists of two pipelines on 3 processors (with id 1, 3 and 4) and one network (with id 2). We actually consider two versions of this test case: in the first version (a) pipeline P^1 is periodic with period $200\,ms$ and end-to-end deadline equal to the period. In the second version (b), the period of the first pipeline is reduced to $30\,ms$ (as in the original specification in [WTVL06]). The full set of parameters is reported in Table 2, where all values are expressed in microseconds. We perform parametric analysis on C_5^1 and C_1^2.

For version (a) we run all tools and we report the regions of schedulability in Figure 4. Once again IMITATOR dominates the other two. Also, MAST dominated RTSCAN. The reason is due to the offset-based analysis methodology used in MAST, which reduces the interference on one task from other tasks belonging to the same pipeline.

For version (b) we run only RTSCAN and MAST, because in the current version we only model constrained deadline systems with IMITATOR. The results

[3] By zooming in the figure, it looks like in some very small areas, the region produced by RTSCAN goes over the region produced by IMITATOR. However, remember that both tools only deal with integer numbers; that small region does not contain any integer point.

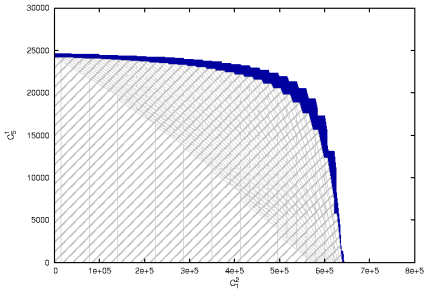

Fig. 5. Schedulability regions for test case 2b, produced by RTSCAN (grey, below) and MAST (dark blue, above)

Table 3. Execution times of the tools

Test Case	RTSCAN	MAST	IMITATOR
1	0.27s	7 s	19min42
2a	0.47s	40min13	2h08
2b	1min11	33min19	–

for version (b) are reported in Figure 5. In this case, MAST dominates RTSCAN. Again, this is due to the fact that MAST implements the offset-based analysis.

6.3 Execution Times

Before looking at the execution times of the three tools in the three different test cases, it is worth to discuss some details about their implementation.

IMITATOR produces a disjunction of convex regions. However, these regions are typically small and disjoints. Moreover, to produce a region, IMITATOR needs to start from a candidate point on which to call *IM*, and then move to close-by regions. One key factor here is how this search is performed. Currently, IMITATOR searches for a candidate point in the neighbourhood of the current region. This is a very general strategy that works for any kind of PSA. However, the particular structure of schedulability problems would probably require an ad-hoc exploration algorithm.

MAST can perform sensitivity analysis on one parameter (called *slack computation* in the tool), using binary search on a possible interval of values. Therefore, to run the experiments, we performed a cycle on all values of one parameter (with a predefined step) and we asked MAST to compute the interval of feasible values for the other parameter.

All experiments have been performed on an Intel Core I7 quad-core processor (800 MHz per processor) with 8 GiB of RAM. The execution times of the tools in the three test cases are reported in Table 3. RTSCAN is the fastest method in all test-cases. In test case 2b, the execution time of RTSCAN is much larger than the one obtained from test case 2a. This is due to the fact that in test case 2b one pipeline has end-to-end deadline greater than the period, and therefore RTSCAN needs to compute many more inequalities (for all points in the hyper-period). Finally, IMITATOR is the slowest of the three and does not scale well with the size of the problem. We observed that the tool spends a few seconds for computing the schedulability region around each point. However, the regions

are quite small, and there are many of them: for example, in test case 2a IMITATOR analysed 257 regions. Also, the tool spends a large amount of time in searching for neighbourhood points. We believe that some improvement in the computation time of IMITATOR can be achieved by coming up with a different exploration strategy more specialised to our problem.

We also evaluated the scalability of RTSCAN with respect to the number of parameters. To do this, we run the tool on test case 2b with a varying number of parameters. The computation time went from 1min11 for $F = 2$ parameters, up to 20min15 for the case of $F = 6$. With $F = 6$, the memory used by our program took a peak utilisation of 7.2 GiB, close to the memory limit of our PC. However, we believe that 6 parameters are sufficient for many practical engineering uses.

7 Conclusions and Future Work

In this paper we presented two different approaches to perform parametric analysis of distributed real-time systems: one based on analytic methods of classic schedulability analysis; the other one based on model checking of PSA. We compared the two approached with classical holistic analysis.

The results are promising, and we plan to extend this work along different directions. Regarding the analytic method, we want to enhance the analysis including static and dynamic offsets, following the approach of [PGH98]. Also, we want to test the scalability of our approach on industrial test-cases.

As of IMITATOR, we plan to improve the algorithm to explore the parameters space: a promising idea is to use the analytic method to find an initial approximation of the feasible space, and then extend the border of the space using PSA.

Acknowledgements

We would like to express our gratitude to Michael González Harbour and Juan M. Rivas, from the Universidad de Cantabria, for their support to installing and using the MAST tool.

The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement No. 246556.

References

AAM06. Yasmina Abdeddaïm, Eugene Asarin, and Oded Maler. Scheduling with timed automata. *Theoretical Computer Science*, 354(2):272–300, 2006. 9

AD94. Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994. 8

AF10. Étienne André and Laurent Fribourg. Behavioral cartography of timed automata. In *RP*, volume 6227 of *Lecture Notes in Computer Science*, pages 76–90. Springer, 2010. 9, 11

AFKS12. Étienne André, Laurent Fribourg, Ulrich Kühne, and Romain Soulat. IMITATOR 2.5: A tool for analyzing robustness in scheduling problems. In *FM*, volume 7436 of *Lecture Notes in Computer Science*, pages 33–36. Springer, 2012. 11

AHV93. Rajeev Alur, Thomas A. Henzinger, and Moshe Y. Vardi. Parametric real-time reasoning. In *STOC*, pages 592–601. ACM, 1993. 8

AM02. Yasmina Adbeddaïm and Oded Maler. Preemptive job-shop scheduling using stopwatch automata. In *TACAS*, volume 2280 of *Lecture Notes in Computer Science*, pages 113–126. Springer-Verlag, 2002. 8, 9

AS13. Étienne André and Romain Soulat. *The Inverse Method*. FOCUS Series in Computer Engineering and Information Technology. ISTE Ltd and John Wiley & Sons Inc., 2013. 2, 3, 9, 11

BB04. Enrico Bini and Giorgio C. Buttazzo. Schedulability analysis of periodic fixed priority systems. *IEEE Transactions on Computers*, 53(11):1462–1473, 2004. 6

BHZ08. Roberto Bagnara, Patricia M. Hill, and Enea Zaffanella. The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. *Science of Computer Programming*, 72(1–2):3–21, 2008. 8

Bin04. Enrico Bini. *The Design Domain of Real-Time Systems*. PhD thesis, Scuola Superiore Sant'Anna, 2004. 2, 3, 5

BLV07. Reinder J. Bril, Johan J. Lukkien, and Wim F. J. Verhaegh. Worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred preemption revisited. In *ECRTS*, pages 269–279. IEEE Computer Society, 2007. 6

CPR08. A. Cimatti, L. Palopoli, and Y. Ramadian. Symbolic computation of schedulability regions using parametric timed automata. In *RTSS*, pages 80–89, 2008. 3

DBBL07. Robert I. Davis, Alan Burns, Reinder J. Bril, and Johan J. Lukkien. Controller area network (can) schedulability analysis: Refuted, revisited and revised. *Real-Time Systems*, 35:239–272, 2007. 2

FLMS12. Laurent Fribourg, David Lesens, Pierre Moro, and Romain Soulat. Robustness analysis for scheduling problems using the inverse method. In *TIME*, pages 73–80. IEEE Computer Society Press, 2012. 2, 3

GHGGPGDM01. M. Gonzalez Harbour, J. J. Gutierrez Garcia, J. C. Palencia Gutierrez, and J. M. Drake Moyano. Mast: Modeling and analysis suite for real time applications. In *ECRTS*, pages 125–134, 2001. 12

GHGPD01. M. González Harbour, J.J. Gutiérrez, J.C. Palencia, and J.M. Drake. MAST: Modeling and analysis suite for real-time applications. In *ECRTS*, 2001. 2

Gru. Grupo de Computadores y Tiempo Real, Universidad de Cantabria. MAST: Modeling and analysis suite for real-time applications. http://mast.unican.es/. 2

HHJ+05. 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 Proceedings* -, 152(2):148 – 166, 2005. 2

LPPR13. Thi Thieu Hoa Le, Luigi Palopoli, Roberto Passerone, and Yusi Ramadian. Timed-automata based schedulability analysis for distributed firm real-time systems: a case study. *International Journal on Software Tools for Technology Transfer*, 15(3):211–228, 2013. 3

PGH98. J. C. Palencia and M. Gonzalez Harbour. Schedulability analysis for tasks with static and dynamic offsets. In *RTSS*, pages 26–37, 1998. 2, 12, 15

RE02. K. Richter and R. Ernst. Event model interfaces for heterogeneous system analysis. In *DATE*, pages 506–513. IEEE Computer Society, 2002. 2

SLS98. Danbing Seto, Dan P. Lehoczky, and Lui Sha. Task period selection and schedulability in real-time systems. In *RTSS*, 1998. 5

SSL+13. Youcheng Sun, Romain Soulat, Giuseppe Lipari, Étienne André, and Laurent Fribourg. Parametric schedulability analysis of fixed priority real-time distributed systems. Research Report LSV-13-03, Laboratoire Spécification et Vérification, ENS Cachan, France, 2013. 7

WTVL06. Ernesto Wandeler, Lothar Thiele, Marcel Verhoef, and Paul Lieverse. System architecture evaluation using modular performance analysis: a case study. *International Journal on Software Tools for Technology Transfer*, 8(6):649–667, 2006. 2, 13

Appendix

A The Inverse Method

We recall the Inverse Method in Algorithm 1.

Algorithm 1: Inverse method $IM(\mathcal{A}, \pi)$

input : PSA $\mathcal{A}$ of initial state s_0, parameter valuation π
output: Constraint K over the parameters

$i \leftarrow 0$; $K_c \leftarrow \text{true}$; $S_{new} \leftarrow \{s_0\}$; $S \leftarrow \{\}$
while true **do**
  while *there are* π*-incompatible states in* S_{new} **do**
    Select a π-incompatible state (l, C) of S_{new} (i.e., s.t. $\pi \not\models C$) ;
    Select a π-incompatible J in $C{\downarrow_U}$ (i.e., s.t. $\pi \not\models J$) ;
    $K_c \leftarrow K_c \wedge \neg J$; $S \leftarrow \bigcup_{j=0}^{i-1} Post^j_{\mathcal{A}(K_c)}(\{s_0\})$; $S_{new} \leftarrow Post_{\mathcal{A}(K_c)}(S)$;
  if $S_{new} \sqsubseteq S$ **then return** $K \leftarrow \bigcap_{(l,C)\in S} C{\downarrow_U}$
  $i \leftarrow i + 1$; $S \leftarrow S \cup S_{new}$; $S_{new} \leftarrow Post_{\mathcal{A}(K_c)}(S)$

One defines $Post^i_{\mathcal{A}(K)}(S)$ as the set of states reachable from a set S of states in exactly i steps under constraint K. We use $Post$ for $Post^1$. We denote by $C{\downarrow_U}$ the constraint over the parameters obtained by projecting C onto the set of parameters, that is after elimination of the clock variables. We say that a set of states S_1 is *included* into a set of states S_2, denoted by $S_1 \sqsubseteq S_2$, if $\forall s : s \in S_1 \Longrightarrow s \in S_2$.

Algorithm 1 uses 4 variables: an integer i measuring the depth of the state space exploration, the current constraint K_c (that is refined by addition of the π-incompatible inequalities), the set S of states explored at previous iterations, and a set S_{new} of states explored at the current iteration i.