A Simulation Methodology for Worst-Case Response Time Estimation of Distributed Real-Time Systems

Soheil Samii, Sergiu Rafiliu, Petru Eles, Zebo Peng
Department of Computer and Information Science
Linköpings universitet, Sweden
{sohsa,serra,petel,zpe}@ida.liu.se

Abstract

In this paper, we propose a simulation-based methodology for worst-case response time estimation of distributed real-time systems. Schedulability analysis produces pessimistic upper bounds on process response times. Consequently, such an analysis can lead to overdesigned systems resulting in unnecessarily increased costs. Simulations, if well conducted, can lead to tight lower bounds on worst-case response times, which can be an essential input at design time. Moreover, such a simulation methodology is very important in situations when the running application or the underlying platform is such that no formal timing analysis is available. Another important application of the proposed simulation environment is the validation of formal analysis approaches, by estimating their degree of pessimism. We have performed such an estimation of pessimism for two response-time analysis approaches for distributed embedded systems based on two of the most important automotive communication protocols: CAN and FlexRay.

1. Introduction and Related Prior Work

Many real-time applications are nowadays implemented on distributed embedded systems comprising a set of communicating *computation nodes* interconnected in a network. Typically, and particularly in automotive embedded systems, the network consists of one or several buses that connect the nodes together. For real-time applications, worst-case behavior is often of interest. Therefore, the system must be analyzed—considering worst-case execution scenarios—to determine whether or not requirements are met. Researchers in the real-time systems community have proposed worst-case timing analysis techniques [11] that more recently have been extended to multiprocessor systems [6, 14, 7, 8, 3] with various scheduling policies for processes, and various communication protocols for the message scheduling on the bus. Due to the high computational complexity of finding exact worst-case response times, the proposed methods are based on heuristics that can analyze large system configurations efficiently. However, the heuristics are *pessimistic;* that is, the obtained response times are upper bounds on the worst-case response times.

A pessimistic analysis can potentially lead to overdesigned and underutilized systems, thus resulting in unnecessarily increased costs. In this context, as we will show, even if worst-case behavior is of interest, simulation can be used as an efficient complement, because it can produce tight lower bounds on worst-case response times. Another typical situation when simulation is the practical method to use is when the running application or the underlying platform is such that no formal timing analysis is available. This can be the case with certain communication protocols, or when the application does not satisfy strict assumptions required by the available timing analysis (e.g., strict periodicity or restricted scheduling policies). An important potential application of simulation, in the context of worst-case behavior, is the validation of a formal analysis approach, with regard to its pessimism. Because a good simulation methodology can produce tight lower bounds on worst-case response times, comparing such a lower bound with the upper bound produced by the analysis provides valuable information on the potential pessimism of the analysis.

Unfortunately, the literature has been meager on the topic of simulation methodologies for worst-case behavior, and on the evaluation of the tightness of different schedulability tests. Bini and Buttazzo [1] studied the performance evaluation of schedulability tests for mono-processor periodic systems without data dependencies between processes. Thaker et al. [13] considered multiprocessor real-time systems and used simulations to estimate the pessimism in different schedulability tests. However, the delays due to the communication of messages were neglected; that is, the authors assumed instant communication of messages. Also, the application model was restricted to linear chains of data dependencies between processes, meaning that any process can have at most one predecessor and at most one successor. Further, the authors assumed constant execution times for each process and only the offset between the initial releases of any pair of processes was assumed to be arbitrary and assigned randomly.

The contribution of this paper is twofold. First, we present a simulation methodology for distributed embedded systems, where our objective is to maximize the response time of a particular process in the application. Our methodology allows the simulator to choose execution times of the jobs intelligently, and to explore the space of execution times efficiently so that the simulator comes close to worst-case scenarios and thus yields large response times. Second, we apply the simulation methodology to estimate the pessimism in response-time analyses for distributed embedded systems, using two of the most important automotive communication protocols: CAN [2] and FlexRay [4].

The remainder of this paper is organized as follows. In the next section, we present the application model and architecture for the systems that we consider. In Section 3, we give an overview of our simulation environment, whereafter, in Section 4, we describe and evaluate our proposed simulation methodology. Further, in Section 5, we present the application of our methodology to pessimism estimation of response-time analyses for CAN- and FlexRay-based systems. Finally, in Section 6, we present our conclusions.

2. Preliminaries

2.1. Application Model

We model a real-time application as a set $\mathbf{\Gamma} = \{G_i : i \in \mathcal{I}\}$ ($\mathcal{I}$ is a finite index set), where each $G_i = (\mathbf{P}_i, \mathbf{E}_i, T_i)$ is a *process graph*—a directed acyclic graph. For a given process graph G_i, the finite nonempty set $\mathbf{P}_i$, indexed by the set $\mathcal{I}_i$, consists of processes that are triggered by an event. We assume that this event occurs periodically with the period T_i, releasing a *job* (i.e., an instance) of each process in $\mathbf{P}_i$. Thus, job n of process $P_{ij} \in \mathbf{P}_i$, denoted $P_{ij}^{(n)}$, is released for execution at time $(n-1)T_i$ for any integer $n > 0$. We also define the *hyperperiod* of $\mathbf{\Gamma}$ as $T_{\mathbf{\Gamma}} = \text{LCM}(\{T_i : i \in \mathcal{I}\})$, where $\text{LCM}(X)$ is the least common multiple of the elements in the finite nonempty set $X \subset \mathbb{Z}^+$. Further, we introduce the set $\mathbf{P}_{\mathbf{\Gamma}} = \cup_{i\in\mathcal{I}}\mathbf{P}_i$ consisting of all processes in the application, and we introduce the set $\mathcal{I}_{\mathbf{\Gamma}} = \cup_{i\in\mathcal{I}}\{(i,j) : j \in \mathcal{I}_i\}$ to index $\mathbf{P}_{\mathbf{\Gamma}}$. Thus, the total number of processes in the application $\mathbf{\Gamma}$ is $|\mathcal{I}_{\mathbf{\Gamma}}| = |\mathbf{P}_{\mathbf{\Gamma}}|$. The set of graph edges $\mathbf{E}_i \subset \mathbf{P}_i \times \mathbf{P}_i$ represents data dependencies between the processes in $\mathbf{P}_i$. For example, an edge $(P_{ij}, P_{ik}) \in \mathbf{E}_i$ indicates that, for any integer $n > 0$, the execution of job $P_{ik}^{(n)}$ cannot start until job $P_{ij}^{(n)}$ has finished executing and the produced output data has been communicated to $P_{ik}^{(n)}$. Finally, a process P_{ij} can have a *relative deadline* D_{ij}, which is the maximum allowed time between the release and the completion of any job $P_{ij}^{(n)}$. Hence, the *absolute deadline* of job $P_{ij}^{(n)}$ is $(n-1)T_i + D_{ij}$. Assuming that the completion time of the job is $t_{ij}^{(n)}$, we define the *response time* of the job as $t_{ij}^{(n)} - (n-1)T_i$ (i.e., the difference in time between completion and release). The *worst-case* response time (WCRT) of a process is the largest possible response-time of any job of that process.

2.2. System Architecture

We consider real-time applications running on distributed embedded systems composed of a set of computation nodes $\mathbf{N}$ connected to a bus. Figure 1 shows an example of such a distributed bus-based system with two nodes $\mathbf{N} = \{N_1, N_2\}$. A node sends and receives data on the bus through its communication controller (denoted CC in Figure 1). The communication controller implements the protocol services according to the protocol specification, and runs independently of the CPU of the node. Figure 1 shows an application consisting of two process graphs, where each process is *mapped* to a computation node. Thus, for a given application $\mathbf{\Gamma}$ and a set of nodes $\mathbf{N}$, we have a mapping function $M_{\mathbf{\Gamma}} : \mathbf{P}_{\mathbf{\Gamma}} \longrightarrow \mathbf{N}$ indicating on what computation node a particular process is executed.

Given a network of nodes and a mapping of processes, we introduce, for each process $P_{ij} \in \mathbf{P}_{\mathbf{\Gamma}}$, a bounded *execution-time space* $\mathbf{C}_{ij} \subset \mathbb{R}^+$ consisting of possible execution times of P_{ij}. We assume that the space of execution times is a closed interval $\mathbf{C}_{ij} = [c_{ij}^{\text{bc}}, c_{ij}^{\text{wc}}]$, where c_{ij}^{bc} and c_{ij}^{wc} are the best-case and worst-case execution times of P_{ij}, respectively. We define the execution-time space of $\mathbf{\Gamma}$ as the Cartesian product of sets $\mathbf{C}_{\mathbf{\Gamma}} = \prod_{(i,j)\in\mathcal{I}_{\mathbf{\Gamma}}} \mathbf{C}_{ij} = \prod_{(i,j)\in\mathcal{I}_{\mathbf{\Gamma}}} \left[c_{ij}^{\text{bc}}, c_{ij}^{\text{wc}}\right]$.

On each computation node runs a real-time kernel responsible for the activation of the processes mapped to the node, while respecting the data dependencies. A released job becomes *ready* for execution when it has received all its inputs from the corresponding jobs of all predecessor processes. When a job finishes, produced messages are sent to the successor processes, which may be mapped to the same or to another node. In the latter case, the message is forwarded to the communication controller of the node. The communication controller broadcasts—according to the communication protocol—the message on the bus.

In this paper, we assume that all processes are event-triggered and that the execution of their jobs on the CPU is scheduled by a priority-based preemptive scheduler[1]. At any point in time, the CPU executes the highest-priority job that is ready for execution. We denote the priority of job $P_{ij}^{(n)}$ at time t with $\pi_{ij}^{(n)}(t) \in \mathbb{N}$, for which a larger value indicates a higher priority. Further, for each edge $(P_{ij}, P_{ik}) \in \mathbf{E}_i$, such that $M_{\mathbf{\Gamma}}(P_{ij}) \neq M_{\mathbf{\Gamma}}(P_{ik})$ (i.e., P_{ij} and P_{ik} execute on different computation nodes), a message m_{ijk} with given communication time is introduced on the bus. The instance of the message that is produced by job $P_{ij}^{(n)}$ is denoted $m_{ijk}^{(n)}$. For each message in the system, the necessary parameters required by the communication protocol are given; for example, for the CAN protocol, fixed and unique message priorities must be provided.

3. Simulation Environment

The simulator is implemented on top of the SystemC [12] discrete-event simulation kernel, and is able to simulate temporal and functional behavior. Figure 2 shows a general overview of the simulation environment. The inputs to the simulator are the application model and system architecture, an *execution-time generator* that is a description of how to choose the variable execution times of the jobs during simulation, C code for the individual processes in the application (only if functional simulation is required), and a *stopping condition* for the simulation. The outputs of the simulator are values related to both the temporal and functional simulation.

The simulator supports various scheduling policies for the processes on each computation node, such as fixed-priority scheduling and earliest-deadline-first scheduling for event-triggered processes, and static cyclic scheduling for time-triggered processes [5]. We also support hierarchical schedulers for the case where time- and event-triggered activities share the same resource [7]. Further, the simulator supports message scheduling on the bus according to several broadcast communication protocols. Currently, we have implemented three protocols common in automotive embedded systems: Time-Triggered Protocol [5], CAN [2], and FlexRay [4].

The execution-time generator, which is addressed in Section 4, is responsible for providing an execution time of each released job $P_{ij}^{(n)}$ in the system. This execution time can be chosen to be any point in $\mathbf{C}_{ij} = \left[c_{ij}^{\text{bc}}, c_{ij}^{\text{wc}}\right]$. The choice of the job execution times affects the response times directly.

[1] We assume a worst-case overhead for the real-time kernel.

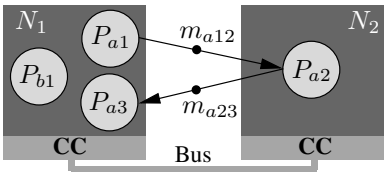

Figure 1. System architecture example

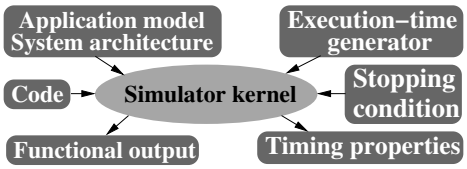

Figure 2. Overview of the simulation environment

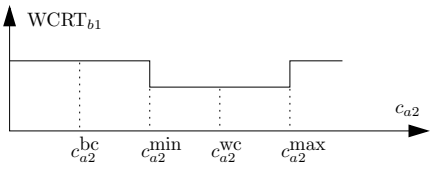

Figure 3. The WCRT of P_{b1} as a function of c_{a2}

Finally, an optional input to the simulator is C code for the processes in the application. We will not further elaborate on this aspect, because, in this paper, we are only interested in temporal simulation.

Definition 1. (Simulation) Let $\mathbf{\Gamma}$ be a real-time application mapped to an architecture, and let t_{sim} denote the amount of simulated time. A *simulation* of $\mathbf{\Gamma}$ means running the system in the time interval $[0, t_{\mathrm{sim}}]$ with given execution times of the jobs that are released in the simulated time interval. Thus, a simulation is determined by a sequence of execution times $\mathbf{c}_{ij}$ for each process P_{ij}, according to $\mathbf{c}_{ij} = \left(c_{ij}^{(n)}\right)_{n=1}^{N_i} = \left(c_{ij}^{(1)}, c_{ij}^{(2)}, \ldots, c_{ij}^{(N_i)}\right)$, where $c_{ij}^{(n)} \in \mathbf{C}_{ij}$ is the execution time of job $P_{ij}^{(n)}$ and $N_i = \lceil t_{\mathrm{sim}}/T_i \rceil$ is the number of job releases of process P_{ij} in the simulated time interval $[0, t_{\mathrm{sim}}]$. The result of a simulation is a response time of each job that finished in the simulated time interval. ∎

As a results of the simulations, we are, for each process, interested in the largest response time of any job of that process. This response time is the estimate of the WCRT of that process.

4. Simulation Methodology

In this section, we focus on the execution-time generator in Figure 2. The choice of job execution times affects the response times directly. It is common knowledge that, in the context of a multiprocessor system, the WCRT of a certain process does not necessarily occur when all jobs of all processes execute for their worst-case execution times. Such *scheduling anomalies* are very common and, therefore, smaller execution times of some jobs or communications can lead to contexts where the response time of another process is maximized [11].

To find the WCRT of a process in the application, we would have to, for all processes, try all possible execution times and all permutations of them. Also, we would have to know the amount of time that needs to be simulated to find the WCRT. In general, none of the two are feasible. The goal of our simulations is to explore the execution-time space in a way that maximizes the response time of a process. We have identified two subproblems that have to be solved to achieve an efficient exploration of the space of process execution times: (1) how to reduce the space of execution times to be explored; and (2) how to generate the next exploration point at a given moment of the simulation process—in other words, what exploration strategy to use.

4.1. Execution-Time Space Reduction

We propose two approaches for choosing a subspace $\mathbf{C}'_{\mathbf{\Gamma}} \subseteq \mathbf{C}_{\mathbf{\Gamma}}$ of execution times to be considered in the simulation. The first approach is to consider only the end points of the execution-time intervals of the individual processes. We refer to this as the *corner-case* values, and introduce the notation $\mathbf{C}_{ij}^{\mathrm{CC}} = \{c_{ij}^{\mathrm{bc}}, c_{ij}^{\mathrm{wc}}\}$ for the two corner-case values of a process $P_{ij} \in \mathbf{P}_{\mathbf{\Gamma}}$. Similarly, with $\mathbf{C}_{\mathbf{\Gamma}}^{\mathrm{CC}}$ we denote the execution-time space of $\mathbf{\Gamma}$ reduced to corner-case execution times—that is, $\mathbf{C}_{\mathbf{\Gamma}}^{\mathrm{CC}} = \prod_{(i,j)\in\mathcal{I}_{\mathbf{\Gamma}}} \mathbf{C}_{ij}^{\mathrm{CC}} = \prod_{(i,j)\in\mathcal{I}_{\mathbf{\Gamma}}} \{c_{ij}^{\mathrm{bc}}, c_{ij}^{\mathrm{wc}}\}$.

The second approach is based on a method for extending the space of corner-case execution times with execution times that are obtained analytically, and that are of interest for the exploration phase. The method, which we refer to as *improved corner-case reduction*, is based on the sensitivity analysis developed by Racu and Ernst [9]. The problem that was studied is the effect that variations of the execution time of a process have on the WCRT of another process in the system. As an example, let us again study the application depicted in Figure 1. We assume that the scheduling policy is based on fixed priorities, ordered according to $\pi_{a1} > \pi_{a3} > \pi_{b1}$. Further, we are interested in variations in the execution time of P_{a2} and the effect on the response time of P_{b1}. If the execution time of P_{a2} is small, then P_{b1} is preempted by P_{a3}, resulting in a certain response time of P_{b1}. On the other hand, if the execution time of P_{a2} is large, then it takes longer time for P_{a3} to become ready for execution, and consequently P_{b1} has time to finish its execution without any preemption from P_{a3}, thus resulting in a smaller response time.

Racu and Ernst [9] have proposed an algorithm that, given an application $\mathbf{\Gamma}$—mapped to a system architecture—and a process $P_{kl} \in \mathbf{P}_{\mathbf{\Gamma}}$, finds a subset of processes $\mathbf{P}'_{\mathbf{\Gamma}} \subset \mathbf{P}_{\mathbf{\Gamma}}$ for which the variation of the execution time of a process $P_{ij} \in \mathbf{P}'_{\mathbf{\Gamma}}$ can *potentially* lead to nonmonotonic behavior of the WCRT of a process P_{kl}. Further, the authors have presented an analysis that points out execution scenarios that lead to such scheduling anomalies for P_{kl}. Figure 3 shows an example of a result of such an analysis for P_{b1}.[2] The figure shows the calculated WCRT of P_{b1} as a function of c_{a2} (the execution time of P_{a2}). It can be seen that the WCRT decreases after the point c_{a2}^{min}, and therefore it is a point of interest in the simulation. The analysis points out an interval of execution times for P_{a2} that all lead to scheduling anomalies for P_{b1}. The figure shows that

[2]The analysis by Racu and Ernst [9] is a heuristic based on classical response-time analysis [6]. Thus, the WCRT in Figure 3 is a pessimistic upper bound on the real WCRT.

scheduling anomalies occur if the jobs of P_{a2} have execution times in $\left[c_{a2}^{\min}, c_{a2}^{\max}\right]$. If the condition $c_{a2}^{\text{bc}} < c_{a2}^{\min} < c_{a2}^{\text{wc}} < c_{a2}^{\max}$ holds, then $\mathbf{C}_{a2}^{\text{ICC}} = \mathbf{C}_{a2}^{\text{CC}} \cup \left\{c_{a2}^{\min}\right\} = \left\{c_{a2}^{\text{bc}}, c_{a2}^{\min}, c_{a2}^{\text{wc}}\right\}$, and otherwise $\mathbf{C}_{a2}^{\text{ICC}} = \mathbf{C}_{a2}^{\text{CC}}$.

In conclusion, given an application $\mathbf{\Gamma}$—mapped to a system architecture—and a process $P_{kl} \in \mathbf{P}_{\mathbf{\Gamma}}$ under investigation, the improved corner-case reduction method consists of two steps. First, we determine the set $\mathbf{P}'_{\mathbf{\Gamma}} \subset \mathbf{P}_{\mathbf{\Gamma}}$ of processes that can potentially lead to nonmonotonic behavior of the WCRT of P_{kl}. Second, we run the analysis for all processes in $\mathbf{P}'_{\mathbf{\Gamma}}$ to find a subset of processes $\mathbf{P}''_{\mathbf{\Gamma}} \subset \mathbf{P}'_{\mathbf{\Gamma}}$ for which the analysis found nonmonotonicities of the WCRT of P_{kl}. Thus, for each $P_{ij} \in \mathbf{P}''_{\mathbf{\Gamma}}$, we have an interval $\left[c_{ij}^{\min}, c_{ij}^{\max}\right]$ that consists of those execution times for which the WCRT function of the process under investigation has nonmonotonic behavior. The reduced execution-time space of $P_{ij} \in \mathbf{P}''_{\mathbf{\Gamma}}$ is now written as $\mathbf{C}_{ij}^{\text{ICC}} = \left(\left\{c_{ij}^{\min}\right\} \cap \mathbf{C}_{ij}\right) \cup \mathbf{C}_{ij}^{\text{CC}}$. For the other processes $P_{ij} \in \mathbf{P}_{\mathbf{\Gamma}} - \mathbf{P}''_{\mathbf{\Gamma}}$, we have $\mathbf{C}_{ij}^{\text{ICC}} = \mathbf{C}_{ij}^{\text{CC}}$. In total, the reduced execution-time space is $\mathbf{C}_{\mathbf{\Gamma}}^{\text{ICC}} = \prod_{(i,j)\in\mathcal{I}_{\mathbf{\Gamma}}} \mathbf{C}_{ij}^{\text{ICC}}$.

4.2. Execution-Time Space Exploration

Let us now assume that we are given a reduced execution-time space $\mathbf{C}'_{\mathbf{\Gamma}} \subseteq \mathbf{C}_{\mathbf{\Gamma}}$ for an application $\mathbf{\Gamma}$ mapped to a system architecture. As discussed in the previous subsection, $\mathbf{C}'_{\mathbf{\Gamma}}$ can be the nonreduced space $\mathbf{C}_{\mathbf{\Gamma}}$, the corner-case reduction $\mathbf{C}_{\mathbf{\Gamma}}^{\text{CC}}$, or the improved corner-case reduction $\mathbf{C}_{\mathbf{\Gamma}}^{\text{ICC}}$. This section discusses two main exploration approaches: *random* exploration and *optimization-based* exploration.

4.2.1. Random Exploration

For the first case for which $\mathbf{C}'_{\mathbf{\Gamma}} = \mathbf{C}_{\mathbf{\Gamma}}$ (no execution-time space reduction), the considered execution-time space of a process $P_{ij} \in \mathbf{P}_{\mathbf{\Gamma}}$ is $\mathbf{C}'_{ij} = \left[c_{ij}^{\text{bc}}, c_{ij}^{\text{wc}}\right]$. At each job release, we choose an execution time in this interval randomly, where each execution time is chosen with equal probability (uniform probability distribution).

For the second case, $\mathbf{C}'_{\mathbf{\Gamma}} = \mathbf{C}_{\mathbf{\Gamma}}^{\text{CC}}$ (corner-case reduction), the execution-time space of process P_{ij} is $\mathbf{C}'_{ij} = \left\{c_{ij}^{\text{bc}}, c_{ij}^{\text{wc}}\right\}$. For each process P_{ij} we specify a discrete probability density function defined as

$$p_{ij}^{\text{CC}}(c) = \begin{cases} p_{\text{wc}}^{\text{CC}} & \text{if } c = c_{ij}^{\text{wc}} \\ 1 - p_{\text{wc}}^{\text{CC}} & \text{if } c = c_{ij}^{\text{bc}} \\ 0 & \text{otherwise.} \end{cases}$$

Thus, for each process P_{ij}, each job is assigned randomly the execution time c_{ij}^{wc} with probability $p_{\text{wc}}^{\text{CC}}$, and the execution time c_{ij}^{bc} with probability $1 - p_{\text{wc}}^{\text{CC}}$. With the customizable parameter $p_{\text{wc}}^{\text{CC}}$, the execution-time space exploration can be driven either towards best-case execution times or worst-case execution times.

For the third case, $\mathbf{C}'_{\mathbf{\Gamma}} = \mathbf{C}_{\mathbf{\Gamma}}^{\text{ICC}}$ (improved corner-case reduction), we consider those processes P_{ij} for which $\mathbf{C}_{ij}^{\text{CC}} \neq \mathbf{C}_{ij}^{\text{ICC}}$. The discrete probability density function for P_{ij} is

$$p_{ij}^{\text{ICC}}(c) = \begin{cases} p_{\text{wc}}^{\text{ICC}} & \text{if } c = c_{ij}^{\text{wc}} \\ p_{\min}^{\text{ICC}} & \text{if } c = c_{ij}^{\min} \\ 1 - p_{\text{wc}}^{\text{ICC}} - p_{\min}^{\text{ICC}} & \text{if } c = c_{ij}^{\text{bc}} \\ 0 & \text{otherwise.} \end{cases}$$

Thus, we choose probabilities for each of the three execution times. For the other processes P_{ij} for which $\mathbf{C}_{ij}^{\text{ICC}} = \mathbf{C}_{ij}^{\text{CC}}$, we have that $p_{ij}^{\text{ICC}} = p_{ij}^{\text{CC}}$ and thus specify the value of $p_{\text{wc}}^{\text{CC}}$ to guide the execution-time space exploration.

4.2.2. Optimization-Based Exploration

The random-based exploration policy presented in Section 4.2.1 is very likely to miss potentially interesting points in the execution-time space. To achieve a more intelligent exploration strategy, we have defined the problem at hand as an optimization problem in which the response time of the process under investigation is the cost function to be maximized. To guide the execution-time space exploration, we have developed optimization strategies based on three meta-heuristics: Simulated Annealing, Tabu Search, and Genetic Algorithms (GAs) [10]. Based on extensive experiments, the GA-based approach turned out to be the most efficient and, therefore, it is the one that we outline here. We have implemented the GA-based heuristic for the cases $\mathbf{C}'_{\mathbf{\Gamma}} = \mathbf{C}_{\mathbf{\Gamma}}^{\text{CC}}$ and $\mathbf{C}'_{\mathbf{\Gamma}} = \mathbf{C}_{\mathbf{\Gamma}}^{\text{ICC}}$. Further, we have tuned the parameters of the heuristic experimentally, to obtain a good performance in terms of quality and runtime.

In the GA approach, each member in a population is a solution consisting of assignments of job execution times, possibly leading to the WCRT. We evaluate a solution by running the simulation for a hyperperiod $T_{\mathbf{\Gamma}}$. A process P_{ij} has thus $T_{\mathbf{\Gamma}}/T_i$ job releases during the time $T_{\mathbf{\Gamma}}$, which means that a solution is an assignment of the execution times $c_{ij}^{(1)}, c_{ij}^{(2)}, \ldots, c_{ij}^{(T_{\mathbf{\Gamma}}/T_i)}$ for all $(i,j) \in \mathcal{I}_{\mathbf{\Gamma}}$. In total, the *solution space* $\mathbf{C}_{\mathbf{\Gamma}}^{(T_{\mathbf{\Gamma}})}$ is defined as the Cartesian product

$$\mathbf{C}_{\mathbf{\Gamma}}^{(T_{\mathbf{\Gamma}})} = \prod_{(i,j)\in\mathcal{I}_{\mathbf{\Gamma}}} \mathbf{C}_{ij}^{T_{\mathbf{\Gamma}}/T_i} = \prod_{(i,j)\in\mathcal{I}_{\mathbf{\Gamma}}} \underbrace{(\mathbf{C}_{ij} \times \cdots \times \mathbf{C}_{ij})}_{T_{\mathbf{\Gamma}}/T_i \text{ times}}.$$

Thus, a solution $\mathbf{c} \in \mathbf{C}_{\mathbf{\Gamma}}^{(T_{\mathbf{\Gamma}})}$ is a tuple of $|\mathcal{I}_{\mathbf{\Gamma}}|$ tuples (one for each process)—that is, $\mathbf{c} = (\mathbf{c}_{ij})_{(i,j)\in\mathcal{I}_{\mathbf{\Gamma}}}$. Each such tuple is a tuple of job execution times—that is, $\mathbf{c}_{ij} = \left(c_{ij}^{(n)}\right)_{n=1}^{T_{\mathbf{\Gamma}}/T_i}$, where $c_{ij}^{(n)}$ is the assigned execution time of job $P_{ij}^{(n)}$. The total number of optimization variables (job execution times in a solution) is thus given by the product $n_{\mathbf{\Gamma}}^{(T_{\mathbf{\Gamma}})} = \prod_{i\in\mathcal{I}} |\mathbf{P}_i| T_{\mathbf{\Gamma}}/T_i$. The cost of each solution $\mathbf{c} \in \mathbf{C}_{\mathbf{\Gamma}}^{(T_{\mathbf{\Gamma}})}$ in a population is defined as the largest response time obtained during simulation with the execution times in $\mathbf{c}$. If no job of the process under investigation finished during the time $T_{\mathbf{\Gamma}}$, then the simulation is run for another hyperperiod $T_{\mathbf{\Gamma}}$ with the same execution times in $\mathbf{c}$.

The size of the initial population is chosen to be $n_{\mathbf{\Gamma}}^{(T_{\mathbf{\Gamma}})}$. However, we have introduced a lower bound of 50 members and an upper bound of 3000 members. The bounds were introduced because a very small population size limits the exploration of the execution-time space, whereas a very high population size results in excessively long runtimes. At each iteration of the genetic algorithm, the simulation is run for all solutions in the current population, whereafter the solutions are sorted according to their quality, given by the obtained response times. In the next step, a new population (the offsprings) is generated by combining the solutions

in the current population, using the crossover and mutation operators [10]. The optimization stops when two criteria are fulfilled: (1) the average value of the cost function for the solutions in the current population has, for 20 consecutive iterations, been at least 95 percent of the current largest response time; and (2) there have been 20 consecutive iterations for which the current largest response time did not change.

4.3. Experimental Evaluation

We have evaluated our simulation methodology through experiments on a set of 81 randomly created synthetic applications consisting of 14 up to 63 processes. The applications were mapped to systems consisting of 2 up to 9 computation nodes that run a fixed-priority preemptive scheduler and communicate on a CAN bus. The individual node utilizations (loads) were set to the values 40, 50, 60, 70, or 80 percent, and the execution times and periods of the processes were produced according to the node utilizations. Further, the data dependencies were generated randomly.

We have compared six different approaches to the execution-time generation, and for each approach we introduce a notation:

- *R,* random exploration of $\mathbf{C}_{\mathbf{\Gamma}}$;
- *R-CC,* random exploration of $\mathbf{C}_{\mathbf{\Gamma}}^{\mathrm{CC}}$;
- *R-ICC,* random exploration of $\mathbf{C}_{\mathbf{\Gamma}}^{\mathrm{ICC}}$;
- *GA-CC,* GA-based exploration of $\mathbf{C}_{\mathbf{\Gamma}}^{\mathrm{CC}}$;
- *GA-ICC,* GA-based exploration of $\mathbf{C}_{\mathbf{\Gamma}}^{\mathrm{ICC}}$; and
- *W,* all execution times are chosen to be the worst-case.

For *R-CC* and *R-ICC*, the probabilities were tuned experimentally and finally chosen as $p_{\mathrm{wc}}^{\mathrm{CC}} = 0.8$ and $p_{\mathrm{wc}}^{\mathrm{ICC}} = p_{\mathrm{min}}^{\mathrm{ICC}} = 0.4$. Regarding the simulation times, we have recorded the runtimes for the two GA heuristics. We let the other simulations, except for *GA-CC,* run for the same amount of time as *GA-ICC.*

For the first comparison, we computed—for each of the approaches and for each application—the ratio $\underline{R}^{\mathcal{S}}/\overline{R}^{\mathcal{A}}$, where $\underline{R}^{\mathcal{S}}$ denotes the lower bound on the WCRT obtained with the simulation approach $\mathcal{S} \in \{R, R\text{-}CC, R\text{-}ICC, GA\text{-}CC, GA\text{-}ICC, W\}$ and $\overline{R}^{\mathcal{A}}$ denotes the upper bound on the WCRT obtained by response-time analysis. Table 1 contains the average ratios obtained by the six different approaches. In the same table, we also show the minimum and maximum ratios that were experienced in the experiments. For the second comparison, we were interested to count the number of times a certain approach $\mathcal{S}$ found the largest response time among the six different simulation approaches (the last column in the table). For example, the *GA-ICC* approach was able to find the largest response time among the six approaches in 97.1 percent of the cases.

Several conclusions can be drawn from the experiments. We can see that random exploration without execution-time space reduction leads to very loose estimates of the WCRT. On the same line, the simulation approach of only assigning worst-case execution times to the jobs is also bad. Further, we observe that the improved corner-case reduction is better than the corner-case reduction method, and that the GA-based execution-time space exploration is better than the random exploration. We conclude that the *GA-ICC* approach yields the best results (largest response times), and is therefore the simulation approach that we used for the pessimism estimation presented in Section 5.

Figure 4 shows the runtime of the *GA-ICC* approach. We show the individual runtimes for different number of nodes (note that, for the applications that were generated, the number of processes grows with the number of nodes). The figure also shows average runtimes that are computed for applications with equal number of nodes. Further, the average runtime for all experiments is 96.2 seconds. It should be noted that all experiments were run on a PC with an AMD ATHLON 64 dual-core CPU running at 2 GHz, 2 GB of memory, and running Linux.

5. Pessimism Estimation

As an application of the simulation environment and methodology presented in Sections 3 and 4, we investigate the performance—in terms of pessimism—of existing response-time analysis techniques for CAN- and FlexRay-based distributed real-time systems. We use a response-time analysis framework [7] using the CAN analysis and the recently developed FlexRay analysis [8]. Let us now define pessimism in the context of response-time analysis.

Definition 2. (Pessimism) Let $\mathbf{\Gamma}$ be a real-time application mapped to an architecture, $P_{ij} \in \mathbf{P}_{\mathbf{\Gamma}}$ denote the process under investigation, and R_{ij} denote the WCRT of process P_{ij}. By applying a response-time analysis $\mathcal{A}$, we obtain an upper bound $\overline{R}_{ij}^{\mathcal{A}}$ on R_{ij}. The *pessimism* in $\overline{R}_{ij}^{\mathcal{A}}$ of the analysis $\mathcal{A}$ is defined as the deviation of $\overline{R}_{ij}^{\mathcal{A}}$ relative to R_{ij}—that is, $\left(\overline{R}_{ij}^{\mathcal{A}} - R_{ij}\right)/R_{ij}$. Further, by using a simulation approach $\mathcal{S}$, we obtain a lower bound $\underline{R}_{ij}^{\mathcal{S}}$ on R_{ij}. We define the *maximum pessimism* as $\left(\overline{R}_{ij}^{\mathcal{A}} - \underline{R}_{ij}^{\mathcal{S}}\right)/\underline{R}_{ij}^{\mathcal{S}}$. ∎

Because of the high computational complexity of finding the WCRT of a process—both in the context of simulation and analysis—we cannot determine the (exact) pessimism, and thus we can only determine the maximum pessimism. Hence, it is important to use a simulation approach that is constructed to find as large response times as possible.

For the pessimism estimation we have created randomly a set of 288 synthetic applications mapped to both CAN- and FlexRay-based systems, respectively. The number of nodes range from 2 to 9, whereas the number of processes range from 10 to 90. Further, the individual node utilizations range from 30 up to 80 percent, and the data dependencies were generated randomly. We simulated each application with the *GA-ICC* approach and computed the maximum pessimism. The results are presented as histograms of pessimism depicted in Figures 5 and 6. For each histogram, we show the maximum pessimism on the horisontal axis, whereas on the vertical axis we show the relative frequency for a certain pessimism—that is, the number of times that a certain pessimism value occured in the experiments, relative to the total number of applications. For example, we can see that, for CAN-based systems (Figure 5), in 20 percent of the cases, both the analysis and the simulation resulted in the WCRT (zero pessimism). Further,

Table 1. Comparison of the six simulation approaches

$\mathcal{S}$	Avg. ratio	Min. ratio	Max. ratio	Frequency
R	77.6%	38.8%	98.3%	0%
R-CC	87.3%	45.1%	100%	30%
R-ICC	87.4%	45.7%	100%	32.9%
GA-CC	88.0%	45.9%	100%	41.4%
GA-ICC	90.5%	49.3%	100%	97.1%
W	83.7%	45.4%	99.8%	0%

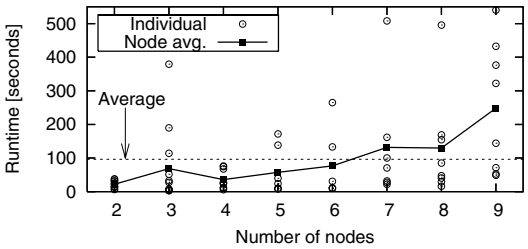

Figure 4. Runtimes for GA-ICC

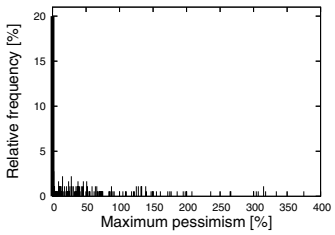

Figure 5. Pessimism histogram (CAN)

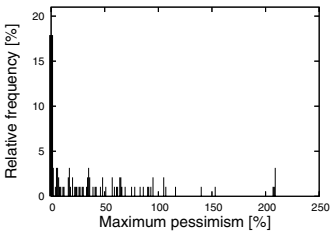

Figure 6. Pessimism histogram (FlexRay)

in 62 percent of the cases, the maximum pessimism was smaller than 50 percent. For FlexRay-based systems (Figure 6), the simulation and the analysis found the WCRT in 18 percent of the cases, whereas the maximum pessimism was smaller than 50 percent for 67 percent of the cases.

Finally, we applied our simulation methodology and the response-time analysis to estimate the pessimism for a cruise-controller application comprising 28 communicating processes mapped to 5 computation nodes. We considered both CAN- and FlexRay-based communication systems. We were interested in the response times of the two processes without successors—that is, the processes producing the control data. For a CAN implementation of the application, we obtained a maximum pessimism of 35.2 and 8.5 percent for the two processes, respectively (the analytical upper bounds on WCRTs are only 35.2 and 8.5 percent, respectively, larger than the lower bounds on WCRTs obtained with simulation). For a FlexRay implementation, we obtained a maximum pessimism of 39.6 and 6.7 percent. Such information is important to the designer, because it allows to appreciate the potential amount of overdesign due to the pessimism of the analysis. The critical processes are in a relatively narrow range of maximum pessimism and, therefore, the actual implementation is tight and cost efficient.

6. Conclusions

In this paper, we proposed a simulation methodology aimed at estimating the WCRTs of processes in distributed real-time applications. We have shown that intelligently reducing the space of execution times to be investigated, combined with an efficient exploration strategy, can considerably improve the quality of produced results. We applied the proposed simulation methodology to evaluate the potential pessimism of two existing schedulability analysis tools for CAN- and FlexRay-based distributed embedded systems. The experiments demonstrate that, by using the proposed approach, the designer can be provided with useful information regarding the bounds on pessimism. The simulation tool can also be used efficiently to obtain tight approximations of the WCRTs in contexts in which no formal response-time analysis has yet been elaborated or such an analysis is not possible due to the nature of the application or execution platform.

References

[1] E. Bini and G. C. Buttazzo. Measuring the performance of schedulability tests. *Journal of Real-Time Systems*, 30(1–2):129–154, 2005.

[2] R. Bosch GmbH. *CAN Specification Version 2.0*. 1991.

[3] R. I. Davis, A. Burns, R. J. Bril, and J. J. Lukkien. Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised. *Journal of Real-Time Systems*, 35(3):239–272, 2007.

[4] FlexRay homepage. `http://www.flexray-group.com`.

[5] H. Kopetz. *Real-Time Systems*. Kluwer Academic, 1997.

[6] J. C. Palencia Gutiérrez and M. González Harbour. Schedulability analysis for tasks with static and dynamic offsets. In *Proceedings of the* 19^{th} *IEEE Real Time Systems Symposium*, pp. 26–37, December 1998.

[7] T. Pop, P. Eles, and Z. Peng. Schedulability analysis for distributed heterogeneous time/event-triggered real-time systems. In *Proceedings of the* 15^{th} *Euromicro Conference on Real-Time Systems*, pp. 257–266, 2003.

[8] T. Pop, P. Pop, P. Eles, Z. Peng, and A. Andrei. Timing analysis of the FlexRay communication protocol. In *Proceedings of the* 18^{th} *Euromicro Conference on Real-Time Systems*, pp. 203–213, 2006.

[9] R. Racu and R. Ernst. Scheduling anomaly detection and optimization for distributed systems with preemptive task-sets. In *Proceedings of the* 12^{th} *IEEE Real-Time and Embedded Technology and Applications Symposium*, pp. 325–334, 2006.

[10] C. R. Reeves, editor. *Modern Heuristic Techniques for Combinatorial Problems*. Blackwell Scientific Publications, 1993.

[11] J. A. Stankovic, M. Spuri, M. Di Natale, and G. C. Buttazzo. Implications of classical scheduling results for real-time systems. *IEEE Computer*, 26(6):16–25, 1995.

[12] SystemC homepage. `http://www.systemc.org`.

[13] G. Thaker, P. Lardieri, D. Krecker, and M. Price. Empirical quantification of pessimism in state-of-the-art scheduling theory techniques for periodic and sporadic DRE tasks. In *Proceedings of the* 10^{th} *IEEE Real-Time and Embedded Technology and Applications Symposium*, pp. 490–499, 2004.

[14] K. Tindell and J. Clark. Holistic schedulability analysis for distributed real-time systems. *Euromicro Jurnal on Microprocessing and Microprogramming (Special Issue on Parallel Embedded Real-Time Systems)*, 40:117–134, 1994.