2009 15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications

Simulation-Based Timing Analysis of Complex Real-Time Systems

Markus Bohlin[1,2], Yue Lu[1], Johan Kraft[1], Per Kreuger[2], and Thomas Nolte[1]
[1]Mälardalen Real-Time Research Centre (MRTC), Västerås, Sweden
[2]Swedish Institute of Computer Science (SICS), Kista, Sweden
markus.bohlin@sics.se

Abstract

This paper presents an efficient best-effort approach for simulation-based timing analysis of complex real-time systems. The method can handle in principle any software design that can be simulated, and is based on controlling simulation input using a simple yet novel hill-climbing algorithm. Unlike previous approaches, the new algorithm directly manipulates simulation parameters such as execution times, arrival jitter and input. An evaluation is presented using six different simulation models, and two other simulation methods as reference: Monte Carlo simulation and MABERA. The new method proposed in this paper was 4-11% more accurate while at the same time 42 times faster, on average, than the reference methods.

1. Introduction

Today, most existing embedded real-time systems have been developed in a traditional code-oriented manner. Many of them are also maintained over extended periods of time, sometimes spanning decades, during which they become larger and more complex due to the iterative changes made as part of the system evolution and maintenance. The increasing complexity makes these systems increasingly hard and expensive to maintain and verify.

The ability to perform timing analysis, using response-time analysis (RTA, [1], [2]) or other means, does not only improve the quality of the system verification, but can also reduce development and maintenance costs significantly as potential timing-related errors can be identified and corrected early. Timing errors can otherwise only be detected in late verification phases, where detected bugs often cause major costs and delays. Moreover, timing errors often only occur under very specific conditions, which are hard to detect using testing.

Sadly, it is not possible to make practical use of RTA on a large quantity of existing industrial software systems, as they violate the assumptions of the method. The disregarded assumptions include task dependencies through communication, data-dependent execution times and priorities, and implicit deadlines and timeouts. Moreover, industrial systems often contain code which is difficult and time consuming to analyze using static analysis, and which requires significant manual effort. Consider the following example from a real system, where a task reads all messages in a message queue and process them accordingly:

```
do {
   msg = receiveMessage(MyMessageQueue);
   process_msg(msg);
} while (msg != NO_MESSAGE);
```

The execution time of this loop naturally depends on the number of received messages. Apart from the obvious problem of determining the maximum run-time queue length, other tasks may also preempt the execution of the loop and refill the queue. When this happens, the number of loop iterations (and thus the execution time of the task) is no longer bounded by the maximum queue size.

One solution to the problem outlined above is to use a more detailed system model. Ideally, the model should describe execution control flow on a code level with respect to resource usage and interaction, e.g., inter-process communication, CPU time and logical resources. Analysing detailed models using techniques such as model checking [3] can for complex industrial systems result in a state-space explosion, which in many cases makes exhaustive analysis infeasible.

Another approach at timing analysis, which avoids the problems associated with state-space explosion, is to use sampling of the state space with simulation-based methods. Drawbacks include that an upper bound on response time cannot be guaranteed, and subsequently less confidence should be put in the results. In this aspect, the approach is closer to testing than formal analysis. However, simulation analysis is sufficient in many cases, and can be far more efficient at finding potential timing problems than system-level testing, the dominating method in industry today.

1533-2306/09 $25.00 © 2009 IEEE
DOI 10.1109/RTCSA.2009.41

Several frameworks already exist for timing simulation of real-time system models, e.g., the commercial tool *VirtualTime* [4] and the academic tool *ARTISST* [5]. These solutions rely on Monte Carlo simulation, which can be described as keeping the highest result from a set of randomized simulations.

In this paper, we show that a hill-climbing algorithm [6] working on a detailed representation of the system-dependent simulation parameters can yield substantially better results than both Monte Carlo simulation, which is the current state-of-practice, and another previously proposed method based on genetic algorithms, *MABERA* [7]. Surprisingly enough, as far as we know, this simple idea has not been tried before. The paper contains the following two main contributions: 1) We present a novel algorithm for manipulating simulation instances, based on the simple idea of hill-climbing with random restarts (HCRR), and 2) we give a thorough experimental evaluation of performance, scaling and convergence of the new algorithm, comparing the results to those obtained from MABERA and Monte Carlo simulation. In the evaluation, we show that the new algorithm is significantly better than previous approaches in identifying extreme response times using a limited number of simulations. This is important, since the number of simulations required directly affects evaluation time and the practical feasibility of the approach.

The paper outline is as follows. Section 2 presents related work and the simulation framework used. Section 3 presents the new approach proposed in this paper, and Section 4 describes a set of case-study models used to evaluate the approach. The evaluation is presented in Section 5, and finally, Section 6 concludes.

2. Best-Effort Response-Time Analysis

Response-time analysis is certainly not something new, and besides the standard approaches such as RTA [1], [2], formal analysis tools like UPPAAL [3], [8] can also be used for exhaustive analysis of software systems, but for models of industrial systems, the state space can grow too large for them to be practically useful.

The use of evolutionary algorithms for different types of test case generation has also been studied for quite some time. In [9], genetic algorithms were used to generate test cases for a software relay system used in electrical networks. The purpose of the genetic algorithm is to provoke high response times for the software, which executed in a simulation environment. Nossal et al [10] describe various extensions of the traditional genetic algorithm [11] to better suit the type of problems in the real-time domain. More recently, Mueller and Wegener [12] gave a comprehensive comparison of static analysis techniques and evolutionary algorithms, with regard to schedulability, for several real-time applications.

In [13], Samii et al aim to find extreme response times for distributed systems by optimizing a set of simulation parameters for models containing temporal attributes and communication. They use a genetic algorithm to explore combinations of task execution times in order to maximize end-to-end response time. Flow of control within tasks is not considered. Their results depend on the method developed by Racu and Ernst [14] for identifying situations where decreased execution times can lead to increased response times. The analysis framework by Kim et al [15] also has a similar basis of temporal task attributes. In [7], we presented MABERA, a meta-heuristic approach for best-effort response-time analysis of models of complex legacy systems using ideas from genetic algorithms [11].

2.1. Simulation of Complex Real-Time Systems

The analysis method presented in this paper is based on the simulator framework *RTSSim* [16], which allows for simulating models describing both the functional and temporal behaviour of tasks. An RTSSim simulation model consists of a set of tasks, sharing a single processor. Each task in RTSSim is a C program, which executes in a "sandbox" environment with similar services and runtime mechanisms as a normal real-time operating system, e.g., task scheduling, inter-process communication (message queues) and synchronization (semaphores). The default scheduling policy of RTSSim is preemptive fixed-priority scheduling and each task has scheduling attributes such as priority, periodicity and offset. It is possible to change these parameters dynamically, in the task model code, to implement a custom scheduling policy.

In RTSSim, time is represented in a discrete manner using an integer simulation clock, which is only advanced explicitly by the tasks in the simulation model, using a special routine, EXECUTE. Calls to this routine models the tasks' consumption of CPU time.

All time-related operations in RTSSim, such as time-outs and activation of time-triggered tasks, are driven by the simulation clock, which makes the simulation result independent of process scheduling and performance of the simulation computer. Each task-switch and IPC event during a simulation is recorded by RTSSim and the resulting trace can later be inspected using a graphical tool. RTSSim also measures response time and execution times for each finished instance of a specified task, and reports the maximum values observed during the simulation. This exact monitoring, together with the explicit simulation clock, guarantees that the measured response time is exact. The simulation framework allows for variations in execution times (for EXECUTE) and task inter-arrival times

(jitter), which is directly controlled by simulator input data, from a simulation optimization algorithm (such as HCRR), or from a random number generator (Monte Carlo simulation). Thus, a simulation in RTSSim is completely deterministic given a specific input, in this paper referred to as a *simulation instance*.

Problem Definition. We can define the problem of best-effort response-time analysis with explicit input as follows. We are given a model of a real-time system, which can be simulated on simulation instances S, consisting of simulator parameters. Let $R(S)$ denote the highest response time measured for the task under analysis in the simulation of instance S. The goal of the problem is then to find a simulation instance S^* that maximizes R, subject to the constraints from the underlying real-time system simulator on S^* outlined in Section 2.1.

3. The Optimization Algorithm

In the rest of this paper, we focus on analysing the response time of a specific given task by varying the simulation instances used as input for the simulator. Analysis of an entire system can be done by performing our analysis several times, once for each task in the system.

We have implemented two different algorithms for comparison with our new method. The first one is traditional Monte Carlo sampling, which is the method most often used for automatic testing. The idea behind Monte Carlo sampling is to simulate a process, such as a real-time system, using a sufficiently large number of random states for the result to approximate the real distribution of process results as close as possible. The other method that we compare our new approach against is MABERA [7].

3.1. Random Restart Hill Climbing

The proposed new optimization algorithm, HCRR, is based on hill climbing using random-restarts. Hill-climbing has the advantage of being one of the simplest meta-heuristics available, and is based on the idea of starting at a random point, and then repeatedly taking small steps pointing upwards (in this paper equivalent to the measured response time) whenever such search directions exist. If no such step exist, a local maximum have been reached.

Advantages of HCRR come from the combination of a strictly local improvement part, which quickly converges to high response times, with diversification mechanisms (jump-back to equal candidates, and full restarts) that are important to avoid local maxima. In contrast, MABERA [7] does not employ such a mechanism, and consequently can get stuck in local maxima. In addition, the local improvement functionality of MABERA is inefficient due to a representation of simulation instances as random number sequences. Monte Carlo search, on the other hand, has no mechanism at all for local improvement, and therefore exhibits unsatisfactory convergence.

```
HCRR(nofsims, m, k, nB, nR)
  curr ← MONTECARLO(m, rnd_inst())
  best ← curr,nofsims ← nofsims − m
  E ← {curr}, nonimp ← 0
  while nofsims > 0
    if nonimp > nR
      curr ← rnd_inst(), E ← {curr}, nonimp ← 0
    else if (nonimp + 1) mod nB = nB
      curr ← random element in E
    next ← NBH(curr, ⌊k · len(curr)⌋)
    SIMULATE(next), nofsims ← nofsims − 1
    if R(next) > R(best) then best ← next
    if R(next) > R(curr)
      curr ← next, E ← {next}, nonimp ← 0
    else
      nonimp ← nonimp + 1
      if R(next) = R(curr) then E ← E ∪ {next}
return best
```

Fig. 1. Hill Climbing with Random Restarts

The implementation of HCRR is given in Fig. 1. The simulation budget is denoted $nofsims$, and $\mathrm{TR}(q)$ denotes the end time of the task under analysis in the simulation instance q when the worst response time occurred. The consumption time point of a simulation input X_i^j of any type (jitter, execution time, or environmental input) is expressed as TM_i^j. $q[X_i^j]$ is the current value of X_i^j in the simulation instance q. A completely random simulation instance can be generated using the call $\mathrm{rnd\_inst}()$.

The HCRR algorithm begins by choosing as starting point the best simulation instance from m randomly selected candidates using the MONTECARLO method. Then, in each iteration, $k \cdot \mathrm{len}(curr)$ random values of the current simulation instance $curr$ (which has $\mathrm{len}(curr)$ input values) used before $\mathrm{TR}(curr)$ are selected and modified using the neighborhood procedure NBH, shown in Fig. 2.

```
NBH(inst, n)
  for k = 1 to n
    X_i^j = random element in inst, TM_i^j < ET(inst)
    V = {lb(X_i) ... ub(X_i)} \ {inst[X_i^j]}
    inst[X_i^j] ← random value in V
```

Fig. 2. Neighborhood procedure

The response time for the task under analysis is measured by running RTSSim using the SIMULATE($next$) call on a neighbor $next$. Modifications suggested by NBH that increase response time are accepted, and changes that decrease response time are rejected. Modifications that

have equal response time are rejected but saved for future reference, as described below.

A pure hill-climbing procedure is susceptible to getting stuck in local maxima, and can therefore exhibit less than satisfactory performance on many problems. In order to avoid convergence to locally maximal areas and to improve the probability of finding a true global maximum, two different diversification mechanisms were implemented. First of all, the algorithm jumps back to a previously encountered, randomly selected simulation instance with an equal response time to the current instance after nB non-improving simulations. This distributes focus over a number of equal instances, which can help in avoiding small local maxima. The second mechanism performs a full restart of HCRR from a random point after nR non-improving simulations. We call nB the *jump-back threshold* and nR the *random-restart threshold*.

4. Case Studies

This section describes two industrial cases and one validation case in the form of simulation models. The models have similar architecture and analysis problems as two industrial real-time applications in use at ABB [17] and Arcticus Systems [18]. Although the simulation models contain relatively few tasks, at most 11, their behavioural complexity is significant due to, e.g., shared variables, sporadic events and dynamic priority changes.

Model 1 (M1) is representing a control system for industrial robots developed by ABB Robotics, which is not possible to analyse using methods such as RTA [19], [2]. This model has previously been used to evaluate MABERA in [7]. Model 2 (M2) is constructed from a test application used by Arcticus Systems [18], which develops the Rubus RTOS used in many vehicular systems. We also use a simplified version of Model 1 for validation (MV). The sole purpose of this model is to investigate how close the response times found by HCRR are to the true worst-case response times derived by RTA.

The scheduling policy is preemptive priority-based scheduling for all models. Model 2 and the validation model both use fixed priorities. Model 1 contains one task that changes priority dynamically.

4.1. Model 1

This model represents a control system for industrial robotics, developed by ABB. The ABB system is quite large, containing around 3 millions lines of code and is not analysable using traditional analytical methods, such as RTA. Model 1 is of much smaller scale, but is designed to include some behavioural mechanisms from the ABB system which RTA can not take into account:

- tasks with intricate dependencies in temporal behaviour due to IPC and shared state variables;
- the use of buffered message queues for IPC, where triggering messages may be delayed;
- tasks that change scheduling priority or periods dynamically, in response to system events.

The modeled system controls a set of electric motors based on periodic sensor readings and aperiodic events. The calculations necessary for a real control system are, however, not included in the model; the model only describes behaviour with a significant impact on the temporal behaviour of the system, such as resource usage (e.g., CPU time), task interactions and important state changes. The details of the model are described in [16].

4.2. Model 2

This model is based on a test application from Arcticus systems, developers of the Rubus RTOS [18] which is used in heavy vehicles. This model uses a pipe-and-filter architecture, where tasks trigger other tasks through trigger ports, forming transactions. The model contains 3 periodic transactions and one interrupt-driven task, in total 11 tasks. The interrupt has a small jitter, while the other transactions are strictly periodic.

This model is less complex than Model 1 in the sense that there exist no shared variables or IPC via message passing which can impact the tasks' timing and functional behaviour. Instead, the tasks have large variations in execution times, which makes the state space of this model very large. For this model, the evaluation focuses on the end-to-end response time of the transaction which contains the tasks with the lowest priority. More details of the model can be found in [20].

4.3. Validation

Simulation-based methods for response-time analysis have in common that the result is not guaranteed to be a safe upper bound on the response time. We therefore constructed a validation model, analysable using RTA, with the purpose to investigate how close the response times given by HCRR are to the worst-case response times derived using RTA. Hence, RTA provides an upper bound on the worst-case response time that the simulation-based results should approach but not exceed. The validation model is based on Model 1, but simplified in that 1) shared state variables have been removed, 2) priority and period is strictly static, and 3) loop bounds have been added manually. As a consequence, the validation model has considerably lower complexity, and exhibits quite different timing properties when compared to Model 1. For instance, the worst-case response time of the CTRL task (which as

in Model 1 is the task under analysis) is only 52 % of the highest response times found for this task in Model 1.

Direct application of RTA yielded a worst-case response time of 5 982. However, after reviewing the results of running HCRR on the model, we realized that a refinement taking into account a rare sporadic event was possible. In this refined model, one of the tasks was split into two separate tasks with different WCET and period time. This refinement of the model had a major impact with respect to RTA, yielding a worst-case response time of 4 432, which is equal to the result found by HCRR. Note that such model refinements are difficult to apply in practice for real industrial systems, as their temporal behaviour are rarely documented in sufficient detail.

5. Experimental Evaluation

This section presents an evaluation of accuracy, convergence and scaling properties of HCRR, using in total six different versions of the models described in Section 4. The experiments were done by running HCRR, a reimplementation of MABERA (MAB) and Monte Carlo (MC) simulation, on the three models previously described.

The goal of the analysis is to find extreme response times for a specific task in the model. The results are, with the exception of Fig. 3, obtained from running 100 samples of each algorithm and test case, each sample being allowed to run 10 000 simulations, in order to get a good comparison for a fixed time length. The simulation budget was considered reasonable due to the convergence of HCRR on our most realistic model (Model 1). The experiments were performed on an Intel Core 2 Duo, 2.33 GHz with 2 GB of RAM.

For MABERA, the population was obtained by scaling the population size of 10 000 used in [7] to reflect the change in number of simulations per sample. The ratio is 81 400 in [7] to 10 000 in this paper. As a result, we use a population size of 1 250, which is 1/8 of the original population size. The same fraction of parents as for the original method is used, which translates to a selection of 12 parents in each generation. For each of these, 104 mutations are generated. In order to ensure that MABERA used exactly 10 000 simulations in total, the original termination threshold was disabled.

For the parameters in HCRR, the jump-back threshold (nB) should be relatively small to spread the search over the set of equal candidate solutions found so far. However, the random restart threshold (nR) should be larger in order not to erase any progress made so far, but small enough to force restart from a local maximum as soon as possible. The fraction k of input values changed in each iteration should provide a good balance between power (larger fractions) and low dimensionality (smaller fractions).

To select the parameters for HCRR, we performed a small number of sequential experiments on Model 1, varying one parameter at a time. For each parameter set, we measured the convergence C as the average best result in any iteration (i.e., simulation) for 20 sample runs, or more formally: $C = \sum_{i=1}^{20} \sum_{j=1}^{S} R_i^j / (20 \cdot S)$ where S is the number of simulations and R_i^j denotes the response time found after j simulations in sample run i. The number of simulations was 500 for nB and k and 3 000 for nR. The parameters giving quickest convergence ($nB = 2$, $nR = 300$, and $k = 0.02$) were then used for all experiments. The results of the experiments are shown in Table 1.

TABLE 1. Parameter selection.

$nB = nR = \infty$		$k = 0.02, nR = \infty$		$k = 0.02, nB = 2$	
k	C	nB	C	nR	C
0.01	7796.76	100	7931.37	1000	8308.11
0.02	**8010.90**	50	7902.86	300	**8312.05**
0.03	7988.83	20	7939.70	100	8304.17
0.04	7976.14	10	7972.72	50	8254.26
0.05	7961.80	7	7992.25		
0.07	7944.69	5	7944.27		
0.10	7761.59	4	8001.89		
0.15	7645.62	3	7919.24		
0.20	7604.48	2	**8024.98**		
0.30	7483.33	1	7944.27		

To show the effects of scaling on the three algorithms, Model 1 is used to create larger systems by instantiating several independent instances of it, thereby creating independent "subsystems" where each subsystem is a complete model as described in Section 4, including tasks, input events, state variables and message queues. The subsystems are completely independent, except that they share the same CPU. Subsystems are time-separated by relative offsets (20 000 time units) and have reassigned priorities to avoid clashes, and the execution time of the tasks were scaled to avoid overload. Scaling factors used were $1.0, 1/1.5, 1/1.8$ and $1/2.2$ for $1, 2, 3$ and 4 subsystems.

5.1. Results

The obtained lower bounds on worst-case response time are illustrated by the following labels:

MC: The traditional Monte Carlo approach to generate simulation instances using random input data.

MAB: The MABERA approach, using a population size of 1 250 of which 12 parents are selected for reproduction, unless stated otherwise. The algorithm is modified to run for a limited number of simulations.

HCRR: The new algorithm based on random restart hill climbing. The algorithm is given in Fig. 1.

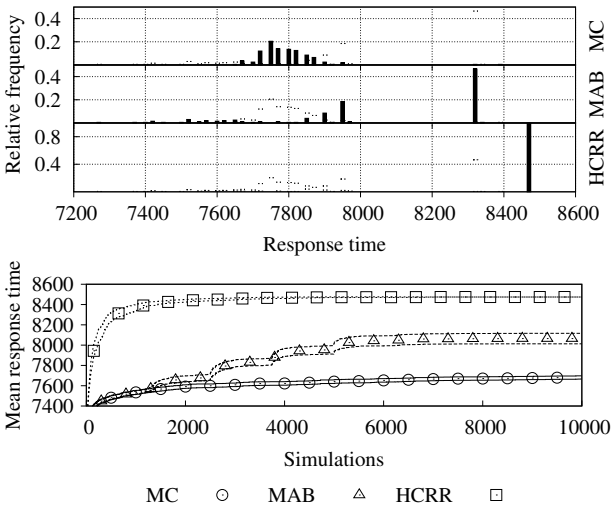

Fig. 3. Final RT distributions and convergence (mean RT and 95% confidence intervals) for model 1.

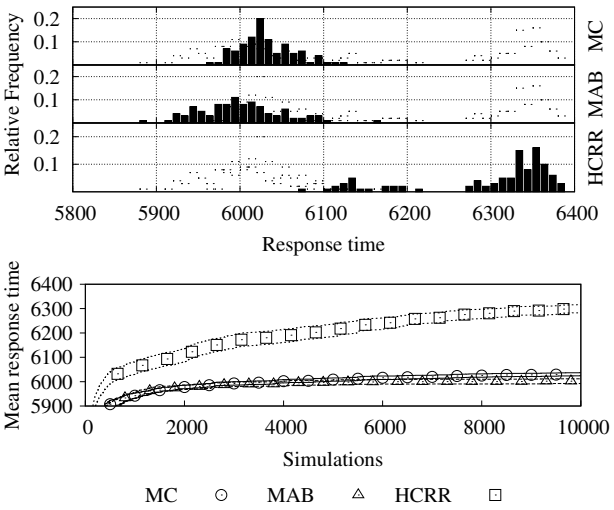

Fig. 4. Final RT distributions and convergence (mean RT and 95% confidence intervals) for model 2.

Fig. 3 shows the results obtained for Model 1 from Section 4.1. The top of the figure contains the response time distributions of the three algorithms, where the MABERA results are taken from [7]. Results were obtained using 200 sample runs for MABERA, 200 runs for MC, and 100 runs for HCRR. For MABERA and MC, each sample required on average 81 400 simulations. Each HCRR sample was allowed 10 000 simulations. The bottom of Fig. 3 shows convergence (mean RT and 95 % confidence intervals), using the standard parameters of 10 000 simulations, for the three algorithms with 100 samples for each algorithm.

The upper part of Fig. 3 shows that HCRR managed to find the highest known response time, 8 474, in all 100 sample runs. The highest response time found by MABERA was 8 349, and this value was only found one single time. The MC approach managed to find a maximum response time of 8 390, which is also found once. Note that HCRR was only allowed approximately 12 % of the number of simulations used by MC and MABERA. If we compare the number of simulations done when the highest known response time was found, HCRR was approximately 1 628 times faster than MABERA and MC. The runtimes for one sample of all algorithms were less than 3 minutes.

Fig. 4 shows the obtained results for Model 2 (Section 4.2) using the standard parameters. In this model, the tasks have large variations in execution times, which makes the state space very large. We can see that HCRR yields a result approximately 5 % higher than what is obtained from the two other methods. Interestingly, it looks like HCRR was still slowly progressing towards higher response times at 10 000 simulations, while both MABERA and MC seems to have converged quite early to a much lower result. For Model 2, all algorithms finished in less than one minute per sample.

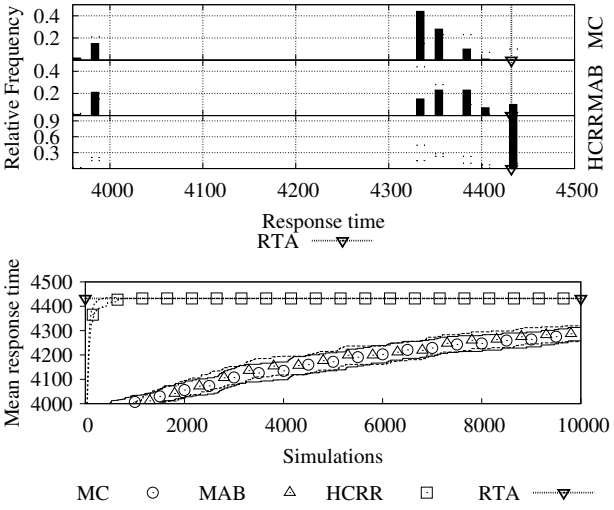

Fig. 5. Final RT distributions and convergence (mean RT and 95% confidence intervals) for the validation model.

In Fig. 5, we can see the results for the validation model described in Section 4.3, again using the standard parameters. In addition, we show the RTA results. Here, HCRR could find a response time of 4 432 in every sample run, which was also confirmed by RTA to be the worst-case response time. As before, the difference between MABERA and MC appears to be quite small. MABERA

found the worst case in a few samples, while MC did not, but it is questionable if the difference is statistically significant. For the validation model, MC took less than 50 seconds, MABERA less than 130 seconds, and HCRR less than 90 seconds for one sample run.

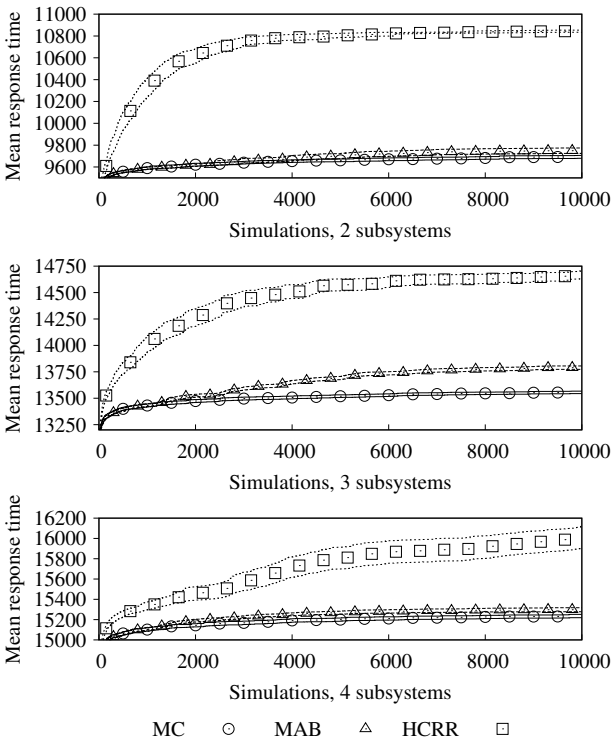

Fig. 6. Convergence (mean RT and 95% confidence intervals) for model 1 using 2-4 subsystems.

Fig. 6 shows how the different methods scale to larger systems, by illustrating the convergence for Model 1 when increasing the model size to 2, 3 and 4 subsystems (model instances). As expected, since the state space increases with number of subsystems, all three algorithms converge slower when system size is increased. For two subsystems, HCRR is consistently better than both MC and MABERA, with all results reported being higher than the maximum result found for both MC and MABERA. The results for 3 and 4 subsystems indicate that the difference between the methods decrease as system size is increased, although HCRR produced on average 4.7 to 11 % higher results than both MC and MABERA. For 4 subsystems, none of the methods appear to have converged. However, during the 10 000 simulations, HCRR progressed more quickly to higher response times than both MC and MABERA. Runtimes for a single sample when having 2 subsystems were below 4, 7 and 5 minutes for MC, MABERA and HCRR, respectively. Sample runtimes were below 5, 10 and 6 minutes for 3 subsystems and below 8, 16 and 10 minutes for 4 subsystems.

TABLE 2. Average end result and point when HCRR passes the second best end result.

	MC	*MABERA*	*HCRR*	*Passes 2nd best*
M1-1	7682	8065	8474	224
M1-2	9693	9750	10844	238
M1-3	13555	13789	14672	521
M1-4	15235	15298	16013	764
M2	6031	6002	6299	634
MV	4286	4288	4432	89

The average end results are summarized in Table 2. The last column also shows the average number of simulations needed for HCRR to obtain the end result of the second best method (using 10 000 simulations). Overall, HCRR reached the second-best result 13 to 112 times faster than the second-best method did. For all models, it took HCRR less than 800 simulations to reach the results of the other methods, which corresponds to less than 1.5 minutes of computation time on the PC used for experiments.

5.2. Average Convergence

To measure average convergence more exactly, we use the relative difference in average response-time results over a time span of d simulations. We say that a method has for practical purposes converged (on average) when $1-\overline{R}^{(k-d)}/\overline{R}^{(k)} \leq \varepsilon$, where $\overline{R}^{(k)}$ is the average response-time result at simulation k for a set of samples. Using this definition, convergence will never be detected before at least d simulations has been performed. In order to measure convergence for the evaluation presented in this paper, d obviously needs to be less than the number of simulations (10 000) performed in each sample. We therefore use $d = 1\,000$ for the convergence comparison. For the tolerance parameter, we chose a value of $\varepsilon = 0.001$. In other words, if the average progress in 1 000 simulations is lower than 0.1%, we declare that the method has converged on average. It should be pointed out that different parameters will give radically different results on convergence, and true convergence is reached and detected only when $\varepsilon = 0$ and d is sufficiently large.

Table 3 summarizes the convergence results obtained with the parameters above, for Model 1 with 1-4 subsystems (M1-1 to M1-4), Model 2 (M2), and the validation model (MV). In general, we can see that HCRR converged to significantly higher response times than MABERA and MC. For the validation model, the only method to converge within 10 000 simulations was HCRR. Overall, the results are mostly consistent with what can be seen in Fig. 3, 4 and 5, but also classified the slow progress for HCRR on M2 in

TABLE 3. Convergence on iteration k to response time $\overline{R}^{(k)}$ for the different methods.

	MC		*MABERA*		*HCRR*	
	k	$\overline{R}^{(k)}$	k	$\overline{R}^{(k)}$	k	$\overline{R}^{(k)}$
M1-1	7632	7670	7356	8062	4090	8466
M1-2	4806	9660	6518	9728	7093	10830
M1-3	3527	13502	7801	13773	5568	14578
M1-4	3410	15175	5104	15271	6948	15881
M2	3656	5997	3552	5991	9556	6295
MV	–	–	–	–	1661	4432

Fig. 4 as convergence. Running the algorithm longer would either yield slightly higher results or confirm convergence.

For M1-4, convergence of HCRR is also detected in iteration 6 948 after a slow progress between simulation 6 000 and 8 000, but as we can see in Fig. 6, more average progress is made after simulation 8 000. Sampling more than 100 runs for M1-4 would most likely even out the slope after simulation 6 000. In any case, HCRR has clearly not converged after 10 000 simulations, and running the algorithm longer would likely yield even higher results.

6. Conclusions

Simulation-based analysis of complex real-time systems has the potential to provide engineers with timing properties of real-time systems not conforming to classical real-time analysis models such as response-time analysis (RTA). In this paper, a new best-effort approach for simulation-based timing analysis has been presented.

In evaluating HCRR, six models of industrial real-time systems were simulated. The results show that HCRR was 4-11% more accurate than the second-best method, and between 13 to 112 times quicker in reaching the end result of the second-best method, on average 42 times. An analysis of convergence indicates that for two cases out of six, even higher response times could be achieved by letting HCRR run longer.

Industrial deployment of HCRR requires a method for extracting simulation models from software systems. A tool for that purpose, MXTC, is currently in development. The execution-time measurements requires context-switch recording with accurate timestamps. This is possible in most real-time operating systems.

Acknowledgement

This work was supported by the Swedish Foundation for Strategic Research via the strategic research centre PROGRESS. We are grateful to Jan Carlsson, Mikael Sjödin, and Björn Lisper for improvement suggestions.

References

[1] M. Joseph and P. Pandya, "Finding Response Times in a Real-Time System," *The Computer Journal (British Computer Society)*, vol. 29, no. 5, pp. 390–395, Oct. 1986.

[2] C. Liu and J. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, 1973.

[3] G. Behrmann, A. David, J. Håkansson, M. Hendriks, K. G. Larsen, P. Pettersson, and W. Yi, "UPPAAL 4.0," in *In Proc. of the Int. Conf. on Quantitative Evaluation of Systems (QEST'06)*, 2006.

[4] "Website of Rapita systems," 2008. [Online]. Available: www.rapitasystems.com

[5] D. Decotigny and I. Puaut, "ARTISST: An Extensible and Modular Simulation Tool for Real-Time Systems," in *Proc. of the IEEE Int. Symp. on Object-Oriented Real-Time Distributed Computing (ISORC '02)*, 2002.

[6] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 2nd ed. Prentice Hall, 2003.

[7] J. Kraft, Y. Lu, C. Norström, and A. Wall, "A Metaheuristic Approach for Best Effort Timing Analysis targeting Complex Legacy Real-Time Systems," in *Proc. of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 08)*, Apr. 2008.

[8] "UPPAAL Website," 2008. [Online]. Available: www.uppaal.com

[9] J. Alander, T. Mantere, G. Moghadampour, and J. Matila, "Searching Protection Relay Response Time Extremes Using Genetic Algorithm — Software Quality by Optimization," in *In Proc. of the Int. Conf. on Advances in Power System Control, Operation and Management (APSCOM-97)*, vol. 1, 1997, pp. 95–99.

[10] R. Nossal and T. M. Galla, "Solving NP-Complete Problems in Real-Time System Design by Multichromosome Genetic Algorithms," in *Proc. of the SIGPLAN 1997 Workshop on Languages, Compilers, and Tools for Real-Time Sys.*, 1997, pp. 68–76.

[11] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Professional, Jan. 1989.

[12] F. Mueller and J. Wegener, "A Comparison of Static Analysis and Evolutionary Testing for the Verification of Timing Constraints," in *Real-Time Systems*, vol. 21. Kluwer, 2001, pp. 268–241.

[13] S. Samii, S. Rafiliu, P. Eles, and Z. Peng, "A Simulation Methodology for Worst-Case Response Time Estimation of Distributed Real-Time Systems," in *Proc. of Design, Automation and Test in Europe (DATE'08)*, vol. 10-14. IEEE, Mar. 2008, pp. 556–561.

[14] R. Racu and R. Ernst, "Scheduling Anomaly Detection and Optimisation for Distributed Systems with Preemptive Task-Sets," in *Proc. of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'06)*. IEEE, Apr. 2006, pp. 325–334.

[15] K. Kim, J. L. Diaz, L. L. Bello, J. M. Lopez, C.-G. Lee, and S. L. Min, "An Exact Stochastic Analysis of Priority-Driven Periodic Real-Time Systems and Its Approximations," *IEEE Transactions on Computers*, vol. 54, no. 11, pp. 1460–1466, 2005.

[16] J. Kraft, "RTSSim – A Simulation Framework for Complex Embedded Systems," Mälardalen U., Tech. Report., Mar. 2009.

[17] "Website of ABB Group." [Online]. Available: www.abb.com

[18] "Website of Arcticus Systems." [Online]. Available: www.arcticus-systems.se

[19] N. Audsley, A. Burns, R. Davis, K. Tindell, and A. Wellings, "Fixed Priority Pre-emptive Scheduling: an Historical Perspective," *Real-Time Systems*, vol. 8, no. 2/3, pp. 129–154, 1995.

[20] M. Bohlin, Y. Lu, J. Kraft, P. Kreuger, and T. Nolte, "Best-effort simulation-based timing analysis using hill-climbing with random restarts," Mälardalen University, Technical Report ISSN 1404-3041 ISRN MDH-MRTC-236/2009-1-SE, June 2009. [Online]. Available: http://www.mrtc.mdh.se/index.php?choice=publications&id=1665