Best-Effort Simulation-Based Timing Analysis using Hill-Climbing with Random Restarts

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

Today, many companies developing real-time systems have no means for accurate timing analysis, as the software violates the assumptions of traditional analytical methods for response-time analysis, and are too complex for exhaustive analysis using e.g. model checking. This paper presents an efficient best-effort approach for timing analysis targeting such systems, where simulations of a detailed system model are controlled by a simple yet novel optimization algorithm, based on hill climbing with random restarts (HCRR). Using a simulation-based approach implies that the result is not guaranteed to be the worst-case response time, but on the other hand, the method can handle in principle any software design. Unlike previous approaches, the new algorithm directly manipulates simulation parameters such as execution times, arrival jitter and input stimulus.

A thorough evaluation is also presented, where HCRR is compared to Monte Carlo simulation (the current state-of-practice) and a previously proposed method. The evaluation is performed using a set of simulation models constructed from existing systems in the robotics and vehicular domain, and shows that for the three models investigated, the proposed method was 4-11% more accurate and vastly more efficient than the other methods. In our evaluation, HCRR found the second-best result on average 42 times faster than the second-best method. For the largest model, HCRR used only 7.6 % of the simulations needed by the second-best method to reach the same result, implying that HCRR scales to larger systems. For the most realistic model, our new method found the highest-known response time 1 628 times faster than the second-best method.

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.

One specific problem with such systems is the risk for introducing timing-related errors. A natural approach to avoid timing-related errors in real-time software would be to use established analytical methods for response-time analysis (RTA, [1], [2]), which provides exact worst-case response times of tasks, given correct worst-case execution times (WCET). Thereby, a system's correctness with respect to temporal requirements can be guaranteed.

The ability to perform timing analysis, using RTA 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 early, during the design of new features, and thereby avoided. Timing errors can otherwise only be detected in late verification phases, where detected bugs often case 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. Such systems might have been initially designed without timing analysis in mind, or development personnel may have introduced violations of RTA during the system evolution, and thereby lost the analyzability.

The authors have observed several issues with respect to RTA in existing industrial/embedded software systems. Some relevant examples are:

- Tasks communicate and trigger other tasks in complex undocumented patterns.
- Task WCET often depends on input.
- The task priority is sometimes changed dynamically.
- Deadlines are not always defined explicitly, but manifest as functional errors when different timeouts expire.

Moreover, some implementations observed in industrial code makes it very hard to perform static WCET analysis. Consider the following example, 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);
```

There are at least two aspects in this example which are hard to analyse statically. First, even though the maximum allowed queue size is usually known, the actual maximum at runtime is not; the developers may have over-dimensioned the queue as a safety margin. Second, and more importantly, other tasks may preempt the execution of the loop and refill the queue. When this happens, the number of loop iterations is no longer bounded by the maximum queue size.

The impact of mechanisms like buffered queues and priority changes can cause very intricate scenarios, where the worst-case is counter-intuitive and extremely hard to predict manually. For instance, in one of the simulation models (Model 1, described in Section 4.1) used for evaluation of our approach, the worst-case response time of the task in focus surprisingly did not occur when the model received the maximum amount of input events. Instead, the worst case turned out to occur when the input events formed a completely different and very intricate pattern. The details of this case is described in [3]. The system model used by analytical methods such as RTA is too simplistic to allow accurate timing analysis of such systems with such behaviors, instead a detailed model is required, where also relevant task behavior can be described.

An example of an industrial real-time system where RTA is not applicable is the control system for industrial robots developed by ABB. This system has a very complex temporal behavior, where some tasks have execution times varying radically due to input-dependent IPC and globally shared state variables, and where tasks may even change scheduling priority. The analytical methods' use of a task-level WCET attribute will in such cases be very pessimistic since the tasks are not independent; there are often dependencies which result in mutual exclusion between different tasks' WCET scenario.

A more detailed system model is therefore necessary for timing analysis of such systems. Ideally, the model should describe the detailed execution control flow on a code level with respect to resource usage and interaction, e.g., inter-process communication, CPU time and logical resources. Simulation-based methods has previously been shown to work well in analysing such large and detailed models, since they only sample the system state space rather than attempting to search it exhaustively. Moreover, simulation-based analysis is far more efficient in finding potential timing problems than system-level testing, the dominating method in industry today. 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 results from a set of randomized simulations.

In this paper, we show that a detailed representation of the simulation parameters in combination with a focused optimization algorithm can yield substantially better results than both Monte Carlo simulation (which is the current state-of-practice) and another previously proposed method, *MABERA* [6]. Specifically, we propose a new approach where key aspects of the system at hand are encoded directly as parameters in the algorithm. We then use a fairly straightforward optimization method based on the well-known hill-climbing algorithm [7]. Surprisingly enough, nobody seems to have tried this before.

The paper contains the following contributions: 1) We give an explicit representation of simulation instances in the form of inputs such as execution time, arrival jitter and external input stimulus is defined, 2) we present a novel algorithm for manipulating simulation parameters, based on the simple idea of hill-climbing with random restarts (HCRR), and 3) 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.

The paper outline is as follows. Section 2 presents related work and the new input representation. 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 [8], [9] can also be used for exhaustive analysis of software systems, but for

industrial-sized models, 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 [10], 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 [11] describe various extensions of the traditional genetic algorithm [12] to better suit the type of problems in the real-time domain. More recently, Mueller and Wegener [13] gave a comprehensive comparison of static analysis techniques and evolutionary algorithms, with regard to schedulability, for several real-time applications.

In [14], 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 [15] for identifying situations where decreased execution times can lead to increased response times. The analysis framework by Kim et al [16] also has a similar basis of temporal task attributes.

In [6], we presented MABERA, a meta-heuristic approach for best-effort response-time analysis of models of complex legacy systems using ideas from genetic algorithms [12].

The approach is based on a simulator using a schedule of random number generator seeds, in turn used to generate random numbers for the parameters of the adhering system model. The seed of the random number generator can be changed at arbitrary time points, and thus provide a crude control mechanism. Due to the seed schedule representation, only mutation is used in the evolutionary algorithm, which inserts randomly selected new seeds at specific simulation time points. The effect of seed switching is that the entire execution trace for the rest of the simulation is changed. Unfortunately this implies that it is not possible to modify a restricted subset of the simulation parameters, for example the execution time for a specific code segment, that might on its own severely affect the response time. For heuristic methods to work well, small changes in a candidate solution should have small but noticeable effects on the objective function. This clearly doesn't hold for MABERA, where a newly inserted seed makes the rest of the simulation behave completely different. Readers can refer to [6] for a more thorough description of the MABERA approach.

2.1. Simulation of Complex Real-Time Systems

The analysis method presented in this paper is based on the simulator framework *RTSSim*, [3], 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 scheduling policy or 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, on top of the default scheduling policy of RTSSim, which is fixed priority preemtive scheduling (FPPS).

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. The response time of tasks is measured whenever the scheduler is invoked, which happens for example at IPC, task switches, EXECUTE statements, operations on semaphores, task activations and when tasks end. This, together with the simulation clock behaviour, guarantees that the measured response time is exact.

The simulation framework allows for three types of selections which are directly controlled by simulator input data.

1) selection of execution times (for EXECUTE),
2) selection of task-arrival jitter, and
3) selection of task control flow.

A simulation in RTSSim is completely deterministic given a specific input, in this paper referred to as a *simulation instance*. Monte Carlo simulation is realized by providing a randomly generated simulation instance.

The models used for the evaluation in this paper were manually designed to contain similar modeling and analysis challenges as the real systems, and contain only the aspects which were considered interesting from a timing analysis perspective. In general, however, a major issue when using simulation for analysis of existing systems is how to obtain the necessary simulation model, which should be a subset of the original program focusing on behaviour of significance for task scheduling, communica-

tion and allocation of logical resources. For many systems, manual modeling would be far too time-consuming and error prone. An approach for automated model extraction are proposed in [17] and a tool implementing this approach is in development, named *MXTC* – Model eXtraction Tool for C. The MXTC tool targets large implementations in C, consisting of millions of lines of code, and is based on a form of program slicing [18]. The model extraction tool was however not yet mature enough for producing real models for the HCRR evaluation in this paper. Due to the size of industrial systems, virtually all "dark corners" of the C programming language will be encountered, which leads to a quite complex tool, which must be very stable, and at the same time scalable to large quantities of code. However, an evaluation using MXTC and HCRR on a large industrial system is planned during 2009.

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 on S^* outlined in Section 2.1.

2.2. Input Representation

A simulation instance is a set of parameters that exactly determine the outcome of a simulation. In this paper, a simulation instance is represented using a set of sequences of integers, where each sequence is associated with either an arrival jitter of a task, an execution time, or an environmental input stimulus[1]. Each value then directly decides a selection of either jitter, execution time, or state in the task control flow. The advantage of this approach is that the direct relationship between representation and model properties makes it possible to locally refine specific aspects of a given simulation instance.

Let $\mathbf{J}_i$ be a sequence of actual jitter values J_i^r experienced by instance r of a task τ_i. We restrict J_i^r to integer values in the interval $[0, \mathrm{ub}(\mathbf{J}_i)]$, where $\mathrm{ub}(\mathbf{J}_i)$ is an upper bound on jitter for task i in units of the smallest measurable time interval (clock ticks) for the target system. Furthermore, let $\mathbf{X}_k$ be a sequence of values for a certain environmental input stimulus or execution time in the simulated program, and X_k^j be the j^{th} such input value. We assume that all stimulus and execution times X_k^j are of integer type and have upper and lower bounds, so that $\mathrm{lb}(\mathbf{X}_k) \leq X_k^j \leq \mathrm{ub}(\mathbf{X}_k)$ for all k, j. Execution times are used only for deciding CPU time consumption of EXECUTE primitives. Bounds on execution times can be analysed using static analysis [19] or estimated through measurements.

A simulation instance S, defining a fully deterministic simulation of the model, is therefore a set

$$\{\mathbf{J}_1, \mathbf{J}_2, \ldots, \mathbf{J}_n, \mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_m\} \quad (1)$$

where n is the number of tasks which have non-zero jitter and m is the number of environmental stimulus and EXECUTE statements. Denote by N_i and M_k the number of values that are used to represent jitter sequence $\mathbf{J}_i$ and input sequence $\mathbf{X}_k$. N_i and M_k can be determined empirically by tracing how many values the simulator uses for each value. In theory, N_i and M_k can be unbounded, and for some long simulations, N_i and M_k may grow to unacceptable levels. In such cases, we suggest to set N_i and M_k to a fixed acceptable level. If there are not enough input values in the sequence, the simulator should report a warning, and start reuse values from the start of the sequence. For the evaluated models in this paper, N_i and M_k were long enough to represent all values used.

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 easily be done by performing our analysis several times, once for each task in the system.

3.1. Random Restart Hill Climbing

Our initial idea was to use a representation of the input parameters to RTSSim, which more directly corresponded to simulation parameters, in a full genetic algorithm [12]. However, initial experiments with the crossover operator, which is the operator most often associated with genetic algorithms, proved unsuccessful and did not show any significant improvement over MABERA. Instead of focusing on the crossover operator, we chose to investigate iterative improvement of a single individual as an alternative. It turned out that hill-climbing [7], augmented with random restarts whenever a local minimum was detected, gave much better performance than MABERA.

The proposed new optimization algorithm, HCRR, is therefore based on hill climbing using random-restarts. Hill-climbing has the advantage of being one of the simplest metaheuristics available, and is based on the idea of starting at a random point, and then repeatedly taking small steps pointing upwards (wrt. the objective function, which in this paper is the measured response

1. Such environmental input stimulus is represented as various number of events generated by environmental task in Model 1 and Validation Model, and various execution time of Software Circuits (SWCs) in Model 2 in Section 4.

time) whenever such search directions exist. If no such steps exist, a local minimum may have been reached. Several techniques for escaping local minima exist (for example Tabu Search [20] and simulated annealing [21]), but a set of limited experiments conducted did not show any significant performance advantage over hill-climbing with random restarts.

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 doesn't employ such a mechanism, and consequently can easily get stuck in local optimas. In addition, the local improvement functionality of MABERA is inefficient in that it is not clearly connected to existing critical features of the solution candidate. Monte Carlo search, on the other hand, has no mechanism at all for local improvement, and therefore exhibits unsatisfactory convergence.

HCRR works by iteratively changing a small portion of the model parameter set, and restarts after a fixed number of non-improving simulations have been tried.

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

Figure 1. Hill Climbing with Random Restarts

The implementation of HCRR is given in Figure 1. Here, the simulation budget is denoted $nofsims$, and $\mathrm{RT}(q)$ 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 stimulus) is expressed as TM_i^j. $q[X_i^j]$ is the current value of X_i^j in the simulation instance q. The function $\mathrm{rnd}(l, u)$ returns a random number between l and u if $l < u$; otherwise, it returns l. A completely random simulation instance can also be generated using the call $\mathrm{rnd\_inst}()$.

HCRR takes a currently best candidate ($best$) as input, which should be a random simulation instance when first called. It then begins by choosing as starting point the best simulation instance from $\min(m, nofsims)$ 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 $\mathrm{RT}(curr)$ are selected and modified using the neighborhood procedure NBH, shown in Figure 2.

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

Figure 2. Neighborhood procedure

The response time for the task under analysis is measured by running RTSSim using the SIMULATE(nb) call on a neighbor nb. 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 optimal areas and to improve the probability of finding a true global maximum, two different diversification mechanisms were implemented. First of all, after nB non-improving iterations, the algorithm jumps back to a previously encountered, randomly selected simulation instance with an equal response time to the current instance. This distributes focus over a number of equal instances, which can help in avoiding small local maxima. The second technique is a common method for avoiding local maxima by restarting the hill-climbing procedure from a random location after a number of iterations. In HCRR, a random restart is performed after a sequence of nR non-improving iterations.

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 [22] and Arcticus Systems [23]. 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 analytical methods such as RTA [24], [2]. This model has previously been used to evaluate MABERA in [6]. Model 2 (M2) is constructed from a test application used by Arcticus Systems [23], which develops the Rubus RTOS used in many vehicular systems. We also use a simplified version of Model 1 for validation (MV), where the code violating the assumptions of RTA has been removed. The 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 used is preemptive priority-based scheduling for all models. Models 2 and 3 use preemptive fixed-priority scheduling. Model 1 uses a preemptive scheduler and mainly static priorities, but contains one task that changes priority dynamically.

4.1. Model 1

This model describes a fictive system designed to be representative for 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 fictive 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 model contains four periodic tasks with the parameters shown in Table 1 (a lower valued priority is more significant).

Table 1. Task parameters for Model 1.

Task	*Priority*	*Period*	*Depends on*
PLAN	5	40000	UI
CTRL	4 or 2	10000 or 20000	PLAN, IO, UI
IO	3	5000	Sensor
DRIVE	1	2000	CRTL, UI

The environmental input stimulus in this problem is a sequence of integers from zero to two, denoting the number of external events that are generated by a sensor, measured in one IO task period. The IO task then sends equally many messages to the CTRL task. The CTRL task may change priority and periodicity in response to two specific events in the model. The PLAN task is responsible for planning the movement of the physical object connected to the motors. The CTRL task calculates control signals for the motors with respect to coordinates sent from the PLAN task and IO events provided by the IO task. The DRIVE task actuates the motors based on the CTRL task output, which impact the execution time of the CTRL task.

The model also describes a user interface (UI) which generate sporadic events which impacts the system behaviour. There are three types of user interface events: START, STOP and GETSTATUS. The START and STOP events makes the system change between two system modes, IDLE and WORKING, with different temporal behaviours. The GETSTATUS event makes PLAN, CTRL and DRIVE send a status message to the user interface, which increases the execution time of those task instances. The task in focus of analysis is the CTRL task.

4.2. Model 2

This model describes a fictive system based on a test application from Arcticus systems, developers of the Rubus RTOS [23] which is used in heavy vehicles. This model uses a pipe-and-filter architecture, and contains 3 periodic transactions and one interrupt-driven task, in total 11 tasks. The inter-arrival time of the interrupt is 5000 simulation time units, with the offset and maximum jitter 500 and 100 simulation time units respectively. Tasks may trigger other tasks using trigger ports. The parameters of tasks and their execution times are given in Table 2.

This model is less complex than the two earlier models in 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 with a periodicity

Table 2. Task parameters for Model 2.

Task	*Period*	*Off.*	*Jitter*	*Prio.*	*Execution*
swcIT_1	5000	500	100	0	[100, 200]
swcIT_2	5000	500	100	0	[100, 200]
swcA_1	5000	0	0	1	[400, 500]
swcA_2	10000	0	0	1	[400, 500]
swcA_3	30000	0	0	1	[400, 500]
swcB_2	10000	0	0	1	[400, 500]
swcB_3	30000	0	0	1	[400, 500]
swcA_et2	10000	0	0	2	[500, 600]
swcA_et3	30000	0	0	2	[500, 600]
swcB_et2	10000	0	0	2	[500, 600]
swcC_et1	30000	0	0	2	[500, 600]

of 30 000 simulation time units, which also contains the tasks with the lowest priority.

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 should provide an upper bound on the worst-case response time, which the simulation-based results should approach but not exceed. The validation model is based on Model 1, but with the following simplifications:

- Selected shared state variables are removed.
- Dynamic changes of priority and period are removed, only static attributes are used.
- Iteration loop bounds are added manually.

As a consequence, the validation model has considerably lower complexity, and exhibit 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.

Due to our extensive knowledge of this specific model, we could deduce that in order to improve the accuracy of the RTA (without being optimistic), the DRIVE task should be modeled as two separate tasks. These two tasks represent two different WCETs of the DRIVE task, depending on a rare sporadic event, where the minimum inter-arrival time is known. However, it is important to realize that such model refinements are hard to apply in practice, for real industrial systems, as the temporal behavior of such systems are rarely documented in detail. This refinement of the model had a major impact with respect to RTA, yielding a worst-case response time of 4432 (refined model) instead of 5982 (without refinement).

5. Experimental Evaluation

This section presents an evaluation of accuracy, convergence and scaling properties of HCRR, using in total 7 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. Table 3 highlights the types of input parameters for the three models, i.e., the decision variables controlled by HCRR, MABERA or the Monte Carlo method.

Table 3. Simulator input parameters for the considered models.

Model	*Input Stimulus*	*Arrival Jitter*	*Execution Time*
Model 1	Variable	Variable	Constant
Model 2	N/A	0	Variable
Validation Model	Variable	Variable	Constant

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 Figure 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 [6] to reflect the change in number of simulations per sample. The ratio is 81 400 in [6] 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 minimum 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 as the average best result in

any iteration (i.e., simulation) for 20 sample runs, or more formally:

$$C = \frac{\sum_{i=1}^{20} \sum_{j=1}^{S} R_i^j}{20 \cdot S}$$

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 3000 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 4.

Table 4. 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. The model setup can be described using the following parameters:

SUBSYSTEMS: The number of subsystems to use, varied between 1 and 4.

CPU_SPEED: The scale factor for all execution times. Let C be the original execution time for a single EXECUTE statement in the model, then $C/\text{CPU\_SPEED}$ is the resulting execution time in the multiply instantiated model.

OFFSET: The relative offsets between subsystems, allowing for different "phasings" between subsystems. Throughout the experiments, a phasing of 20000 time units has been used.

To avoid priority clashes, new priorities are assigned using the formula $P^n = 10P^o + I$, where P^n is the new priority, P^o is the old priority, and I is the subsystem index. For CPU_SPEED we use factors of 1.0, 1.5, 1.8 and 2.2 when having 1, 2, 3 and 4 subsystem instances respectively.

5.1. Timing 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 Figure 1.

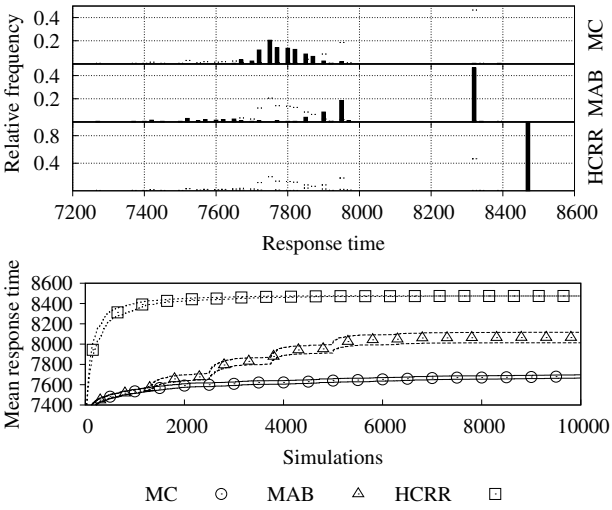

Figure 3. Final RT distributions and convergence (mean RT and 95% confidence intervals) for model 1.

Figure 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 [6]. 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 Figure 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 Figure 3 shows that HCRR managed to find the highest known response time, 8474, in all 100 sample runs. The highest response time found by MABERA was 8349, and this value was only found one single time. The MC approach managed to find a maximum response time of 8390, 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.

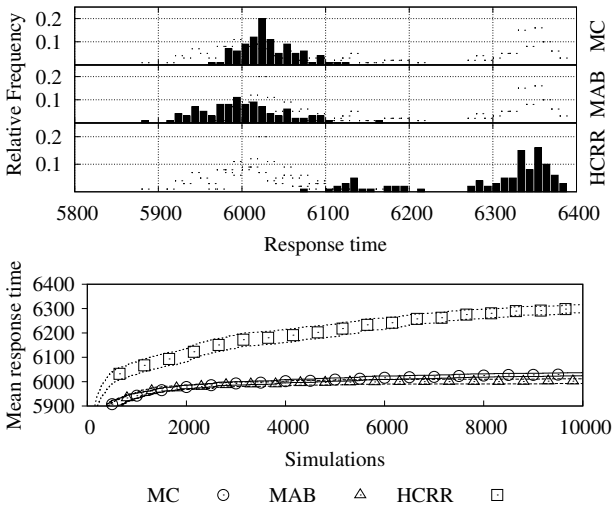

Figure 4. Final RT distributions and convergence (mean RT and 95% confidence intervals) for model 2.

Figure 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.

In Figure 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 4432 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.

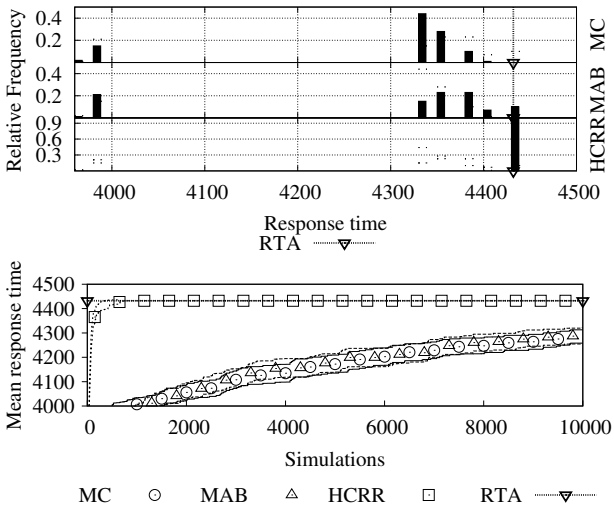

Figure 5. Final RT distributions and convergence (mean RT and 95% confidence intervals) for the validation model.

Figure 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, neither 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. Sample runtimes were below 5, 10 and 6 minutes for 3 subsystems and below 8, 16 and 10 minutes for 4 subsystems.

Table 5. 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

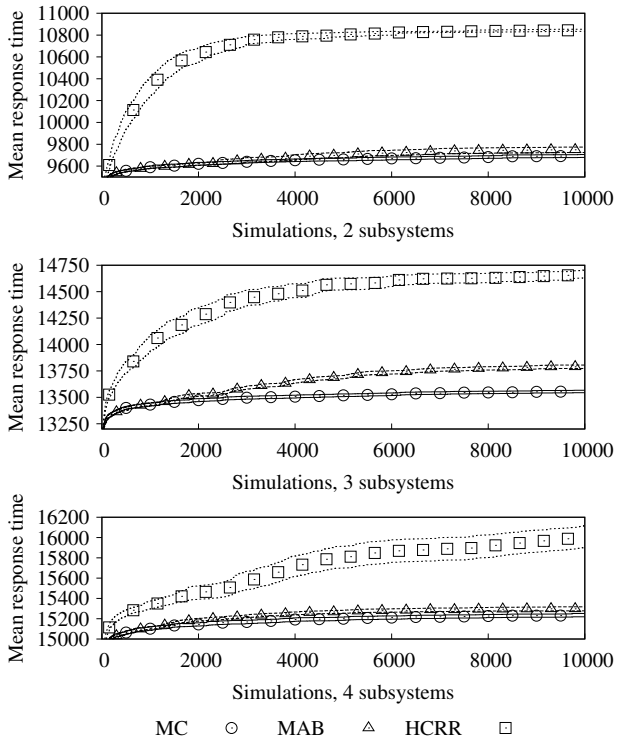

Figure 6. Convergence (mean RT and 95% confidence intervals) for model 1 using 2-4 subsystems.

The average end results are summarized in Table 5. 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). As we can see, HCRR reached the second-best result 13 to 112 times faster than the second-best method did. For all tried models, HCRR on average outperformed the other methods in less than 800 simulations, which corresponds to less than 1.5 minutes of computation time on the PC used for the 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

$$1 - \overline{R}^{(k-d)} / \overline{R}^{(k)} \quad \leq \quad \varepsilon \qquad (2)$$

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 6. 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

Table 6 summarizes the convergence results, obtained from Equation. (2) 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 Figure 3, 4 and 5, but also classified the slow average progress for HCRR on M2 in Figure 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 6948 after a slow progress between simulation 6000 and 8000, but as we can see in Figure 6, more average progress is made after simulation 8000. Sampling more than 100 runs for M1-4 would most likely even out the slope after simulation 6000. 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, and the new algorithm, based on Hill Climbing with Random Restarts (HCRR), is shown in our evaluation to find more

accurate worst-case response time faster than alternative methods such as MABERA and Monte Carlo simulation.

In evaluating HCRR, three models of industrial real-time systems have been simulated, and 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. In one case, HCRR was 1 628 times quicker in finding its more accurate result than the second-best method. An analysis of convergence indicate that for two cases out of six, even higher respones times could be achieved by letting HCRR run longer.

Industrial deployment of HCRR requires an efficient method for extracting simulation models from complex software systems. A tool for that purpose, MXTC, is currently in development. This uses mainly static analysis, but also measurements in order to obtain execution-time data for the model. The simulation model analyzed by HCRR could however use data from WCET analysis tools as well, for supported hardware platforms. 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 comments and 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] J. Kraft, "RTSSim – A Simulation Framework for Complex Embedded Systems," Mälardalen U., Tech. Report., Mar. 2009.

[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] 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.

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

[8] 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.

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

[10] 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.

[11] 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.

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

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] J. Andersson, J. Huselius, C. Norström, and A. Wall, "Extracting Simulation Models from Complex Embedded Real-Time Systems," in *Proc. of the Int. Conf. on Software Engineering Advances, ICSEA'06*. IEEE, Oct. 2006.

[18] M. Weiser, "Program Slicing," in *Proc. of the Int. Conf. on Software Engineering (ICSE'81)*. IEEE Press, 1981, pp. 439–449.

[19] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström, "The Worst-Case Execution-Time Problem—Overview of Methods and Survey of Tools," *Trans. on Embedded Computing Sys.*, vol. 7, no. 3, pp. 1–53, 2008.

[20] F. Glover and M. Laguna, "Tabu Search," in *Modern Heuristic Techniques for Combinatorial Optimization*, C. R. Reeves, Ed. McGraw-Hill, 1995, ch. 3, pp. 70–150.

[21] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by Simulated Annealing," *Science*, vol. 220, pp. 671–680, 1983.

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

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

[24] 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.