Making an ALARP Decision of Sufficient Testing

Mahnaz Malekzadeh[1] and Iain Bate[1,2]
[1]Mälardalen Real-Time Research Centre, Mälardalen University, Västerås, Sweden
[2]Department of Computer Science, University of York, York, UK
email: mahnaz.malekzadeh@mdh.se and iain.bate@york.ac.uk

***Abstract*—*ALARP* is an important concept in many safety standards. It helps in making a decision about how tolerable a risk is. A tolerable risk should be reduced to a point that is *As Low As Reasonably Practicable* (ALARP) which implies further risk-reduction is grossly inappropriate compared to the benefit attained. To date work has considered the process, safety arguments, and influencing factors of how to make an ALARP decision but not shown how to make a quantified judgement for it. In this paper a method for making an ALARP judgement decision is proposed in the context of testing the worst-case timing properties of systems. The method is based around a *convergence algorithm* that informs the tester when it is believed that testing for longer will not reveal sufficiently important new findings, i.e. any significant increase in observed worst-case timing needs a disproportionate amount of testing time.**

I. Introduction

Growing technology, continuous environmental change and public safety concerns make safety analysis of safety-critical systems an important necessary activity. According to the *As Low As Reasonably Practicable (ALARP)* principle, risk-tolerability depends on practicability of further risk-reduction which is a cost-benefit argument. The latter means if the cost of the risk-reduction outweighs the gains to be achieved, there is no need to undertake it provided that the risk is reduced until a tolerable point. The principle of ALARP is an underpinning concept in most safety standards. The first definition of ALARP is "Provision and maintenance of plant and systems of work that are, so far as is reasonably practicable, safe and without risks to health" from the Health and Safety at Work Act in the UK [1] which contained the definition to give safety a legal basis to the concept. Work associated with ALARP has been carried out to define suitable processes [2] and safety arguments [3]. As part of these work, definitions have been produced for what it means for evidence to conform to ALARP, specifically how to make a decision to stop testing a system.

To the best of our knowledge, no previous work has proposed methods that make this decision in a quantified way. This gap in knowledge is addressed in this paper for the important problem of the worst-case timing characteristics of Real-Time Systems (RTS) [4]. Currently there are two principal techniques for understanding these characteristics, static analysis and measurement-based approaches. Measurement-based techniques tend to be used most as static analysis relies on *Worst-Case Execution Times (WCET)* which are expensive to apply if the pessimism is to be reduced to an appropriate level [5]; and *Worst-Case Response Time (WCRT)* analysis is difficult to apply for systems with overheads and complex behaviours, e.g. inter-task dependencies [6].

The principal challenge addressed in this paper is knowing when to stop testing the RTS as no significant new information will be determined without clairvoyance. There is a Measurement-Based Probabilistic Timing Analysis (MBPTA) approach proposed in [7] and [8] to find a tight upper bound for WCET. It collects test data of running an application on target platform, then continuously checks data distributions for convergence to derive probabilities for execution times based on Extreme Value Theory (EVT) [9]. The contributions of this paper are similar to them but have a different basis as follows:

- A convergence algorithm is presented that combines multiple sources of information
- The algorithm has been designed and evaluated with a specific focus on ALARP
- The work has been evaluated in comparison with a *ground truth* to show conformance with ALARP as well as the needs of RTS

In this context, reliable is defined as the algorithm provides valid ALARP decisions across a wide range of task sets. The algorithm makes no assumption concerning the subsequent analysis that the testing information gained is used for. The two main techniques such testing information is used for are *High Watermark (HWM)*, i.e. the *Maximum Observed Response Time (MORT)*, or statistical techniques [10], [11]. The paper contains two contributions. Firstly, the convergence algorithm which has two principal components: whether the MORT is increasing at a sufficiently fast rate, and whether the distribution of response times being measured is indicating if new behaviours are being observed. The intuition behind the algorithm is if the MORT is not increasing and the nature of the response times is not changing, then this implies more testing is not going to reveal further useful information. This is then balanced with the principle of ALARP through the use of appropriate timescales. A separate issue, not covered here, is whether a different testing method would discover new areas of the software and therefore whether the work needs to be combined with suitable coverage metrics. Secondly, an evaluation showing that it reliably decides when sufficient measurements have been made in compliance with the principle of ALARP.

To evaluate the approach, a task set simulator is used. The reason this approach is taken is it allows a ground truth to be established and it allows careful control of the task set characteristics, including complexity. Two ground truths are available for comparison: static analysis which in this particular situation gives an exact safe result [4], and a HWM but with significantly longer simulation. Longer simulation is possible due to the nature of the simulator used, however such increased testing would be prohibitively expensive in a real system.

The structure of the paper is as follows: Section II describes the background and motivation of the work. In Section III the proposed convergence algorithm is presented and Section IV shows the results and evaluates the algorithm. The conclusion is expressed in Section V.

II. Background and Motivation

A. System Model

The system model comprises a set of applications Θ, indexed by ζ_Θ running on execution platform. An application $\Theta_i \in \Theta$ (i $\in \zeta_\Theta$) is modeled as a directed acyclic graph (A_i, B_i), where nodes A_i, indexed by ζ_i, show the system tasks and graph edges $B_i \subset A_i \times A_i$ represent data dependencies between tasks. The set $A_\Theta = \bigcup_{i \in \zeta_\Theta} A_i$ represents all the applications' tasks. Each application $\Theta_i \in \Theta$ has a release period h_i. A job of each task in the application is released for execution at time $(q-1)h_i$. Job q of task τ_{ij} is denoted by τ_{ij}^q. It is released for execution at time $(q-1)h_i$.

The tasks are executed based on the Deadline Monotonic Scheduling Theory (DMST) [12] that implies the shorter the task deadline, the higher priority is assigned to the associated jobs to that task. The executing job can be preempted at any time. By preemption, tasks can interrupt one another so the highest priority task is always the one being executed which means the executing task is halted until the higher priority task runs until completion. Then the halted task resumes execution. Time difference between completion and release time of a job is called its *response time*. If job τ_{ij}^q finishes at time *t*, then its response time is $r_{ij}^q = t - (q-1)h_i$.

B. Typical Approaches for WCET Estimation and WCRT Analysis

Response-Time Analysis (RTA) techniques are traditionally based on simplified assumptions of systems. They compute an absolute WCRT provided that the load on system is bounded and known and exact WCET of each task is determined to judge about system's timing requirements. Such a *deterministic* RTA does not apply to a real system with complex behavior in which not only the tasks possess complex control flow behaviour e.g. due to dynamic calls and dynamic jumps [13] but also have explicit and implicit dependencies, e.g. complex transactions in an engine control systems [14] and global state shared variables in robots' control system [15] respectively.

Moreover, the execution time of individual instructions may significantly change by several orders of magnitude depending on the state of the processor in which they are executed [13]. This causes the execution time of one task heavily depend on the execution state produced by another task. Such complicated temporal and execution dependencies between tasks are difficult to be addressed, making an exact WCET of RTA a difficult task.

Furthermore, the stochastic nature of fault tolerant systems, algorithms with various computation times and modern processors with pipeline, cache and branch-prediction features have lead to researchers investigating probabilistic techniques [16]. Consequently, a RTA framework that is not based on abstract system model and exact WCET estimation seems essential to be developed. Therefore Lu et al. have developed a statistical RTA technique that does not rely on exact estimates of system parameters such as WCET [17].

Due to the effect of false WCET assumptions on WCRT calculation, developers compensate for such wrong assumptions like what is done in [18] and [19] to compensate for cache-related timing effects from preemption and caches shared between cores. They also propose an optimistic *estimate* WCET and a pessimistic *upper bound*.

As stated above, determining the exact WCET is not generally feasible for complex problems and real systems. Typically, WCRT of a task set is calculated based on WCET of each task that finally reveals if all the tasks in system meet their deadlines.

There are two main approaches *Dynamic* (measurement-based) and *static* to determine WCET of a real-time system [20].

1) Static Analysis Approaches: Static approach tries to determine the longest path of the program. Static analysis may result in an unacceptable level of pessimism as the processor architecture gets complex to be predicted due to some advanced features like cache, branch prediction and pipelines. Static analysis is also based on an abstract model of processor timing and may fail to capture effects inhabited in a real system.

2) Measurement-Based Approaches: As an alternative to static analysis, dynamic analysis approach is proposed to record the longest execution time by running the code under exhaustive test conditions. Although, dynamic analysis approaches deal with the real system, it may fail to produce test cases that lead to worst case.

An ideal approach would satisfy two criteria. First, the probability of WCET underestimation has to be sufficiently small resulting to sufficient confidence when claiming that a system meets timing requirements. A noticeable underestimation may incorrectly conform that the system has met deadlines. Second, grossly overestimation should be prohibited due to wasting processors' resources.

Most of the dynamic analysis approaches underestimate the WCET with an unknown probability. They are split into two categories as follows: HWM and probabilistic.

- *HWM* uses the longest execution time observed in testing to estimate the WCET. It is not feasible to determine either the degree or likelihood of underestimation.
- *Probabilistic* where a distribution of each task's observed execution times is formed. Hybrid versions of this analysis benefit from both dynamic and static approaches. Here, the code is divided into *blocks* and the execution times of each block are observed running a real system rather than a processor model. However, the worst-case

effect that is observed locally is combined using static analysis.

C. Measurement-Based Approaches Related to ALARP

There is no other work on methods that make ALARP decision in a quantified way which is addressed in this paper for the worst-case timing characteristics of RTS. However, we notice a similar work on MBPTA to find a tight upper bound for WCET that is proposed in [7]. Basically, MBPTA collects observations of end-to-end runs of an application executed on target platform to derive probabilities for execution times.

There are some efforts based on EVT for WCET estimation that are based on simplified scenarios rather than realistic assumptions of statistical properties of the hardware [21], so not being applicable by industry. Conversely, Cucu-Grosjean et al. propose a MBPTA technique based on a sound application of EVT for multi-path, therefore realistic programs. The results show the estimated *probabilistic WCET (pWCET)* by MBPTA based on EVT in their proposed approach provides less than 15% pessimism compared to static Probabilistic Timing Analysis (PTA). The latter indicates tightest pWCET estimates by PTA with feasibly low observations (650 runs) for all the benchmarks. The authors of [7] also introduced the notion of correct application of EVT by two hypotheses: independence and identical distribution. The former means two random variables *X* and *Y* describe two events such that the occurrence of one event does not have any affect of the occurrence of the other and the latter indicates each random variable in a sequence has the equal probability distribution to the others. The main steps of their approach is as follows:

- *Collecting* execution time observations by running the program under analysis. In each round, N_{delta} further observations are gathered to be included in the next step analysis in case of necessity. In each round, it is also checked whether the data is independent and identically distributed (i.i.d tests) as a prerequisite to EVT by the run-test [22] and the two-sample Kolmogorov-Smirnov (KS) test [23] respectively. The thresholds (values of α) that are needed for the tests to be passed are tuned based on previous experience of these tests.
- *Grouping* that is based on the previously collected observations. It picks HWM values within randomly formed block of data since MBPTA uses EVT to upper-bound the probability of occurrence of the longest execution times.
- *Fitting* finds out if the maximum of execution times resulted by grouping matches one of the EVT distributions: Gumdel, Frechet or Weibull. Among them, Gumdel has shown to fit well the problem of WCET estimation. For this, authors use the Exponential Tail (ET) test [24] that needs to fall in an expected range. Otherwise the hypothesis that the data matches a Gumbel distribution is rejected.
- *Convergence* compares the two successive distributions from current and previous rounds by looking into Continues Rank Probability Score (CRPS) metric. CRPS below a given threshold indicates the current EVT distribution converges to the real WCET distribution. This means no more observation is needed.
- *Tail extension* computes the WCET estimate associated to any exceedance probability threshold by the resulting EVT distribution.

The work in [8] also constructs the basis for safety argumentation towards Critical Real-Time Embedded Systems (CRTES) certification. Although this work seems similar to ours in terms of using a convergence algorithm for WCET estimation, there have some important differences as follows: Here, the convergence algorithm is proposed for making a quantified ALARP decision while in above work, they just reason about finding an upper WCET estimation. We also use HWM to increase scalability and reduce the cost of expensive statistical test that is used.

D. Sufficiency of Test Data

Evidence supporting a safety goal has to be both appropriate and sufficient. There are three characteristics to show the sufficiency of the evidence [3]: *relevance, coverage and independence*. Relevance means the evidence to which extent directly addresses the safety goal. It provides developers an understanding of the role the Software Safety Evidence (SSE) plays. SSE can be the result of WCET analysis that is the most relevant direct evidence implying a Software Safety Requirement (SSR) is met. Coverage points out to which scale the evidence addresses the safety goal that is often used with respect to testing. This indicates the higher the test coverage, the greater the confidence in the achieved results. For instance, availability of resources e.g. input-output buffers and device registers could be a SSR. If a software component depends on availability of four resources and some analysis techniques provide evidence for availability of two resources, the coverage of SSE is 50%. Finally, independence implies to which extent it follows various approaches fulfilling evidence requirements. It is feasible to demonstrate what the evidence shows and which level of confidence is attained by the evidence.

E. Motivational Example

While testing the RTS to conclude evidence for system safety, we are interested in perceiving if any significant new information will be achieved by proceeding with further tests. However, due to the high cost of testing, we would prefer to assess the history of test data to predict future behaviour of the system. This means if it turns that test data has recently showed getting stabilized for a specific period of time, it is anticipated to show similar behaviour further which implies gain of more testing disproportionates huge testing cost. In the context of the system model explained in section II-A, the ideal stopping point can be found by continually examining the history of task's Response Times (RT). We propose HWM and statistics for data analysis and here exemplify how these two help to examine the history of data for an ALARP decision.

At each point in simulation time, HWM shows current MORT and statistics test examines if test data is significantly changing by looking into probability distributions of response times over two consecutive time intervals. Our statistics test is based on *Kullback-Leibler DIVergence (KL DIV)* test [25], [26] that is a non-symmetric measure of difference between two probability distributions *P* and *Q* where *P* presents “true” distribution of data and *Q* is typically a theory, model or approximation of *P*. While traditional inferential statistics show if there is a significant difference between two distributions of data, KL DIV is capable of showing the presence of such a difference either if it is significant or inconsiderable. Consequently, in such a case the *null hypothesis* is a default proposition presenting the absence of difference between two distributions or if the distributions are identical, i.e. KL DIV = 0 . So, the smaller KL DIV value, the closer the test gets to the null hypothesis.

Figure 1 shows the timing behavior of an arbitrary task examined by HWM and KL DIV tests over the simulation time. The horizontal axis shows the number of tests over time (called testing time) while the vertical axis presents KL DIV values at left and MORT at right. In order to run the tests, the simulator periodically outputs response times and two successive of such outputs undergo the tests in each analysis round. Figure 1 (a) zoomed in the dashed gray area in the main plot where testing time is in range [0 1200] and KL DIV in [0 $8{*}10^{-6}$]. It can be seen that KL DIV converges within the first 1200 testing time with MORT equal to 38185. The simulation shows there is in fact one further increase in the MORT, however the increase in MORT is less than 1% and the increase occurs after a lot more testing (150 times more testing). Therefore based on the principle of ALARP, testing could be halted after 1200 testing time units.

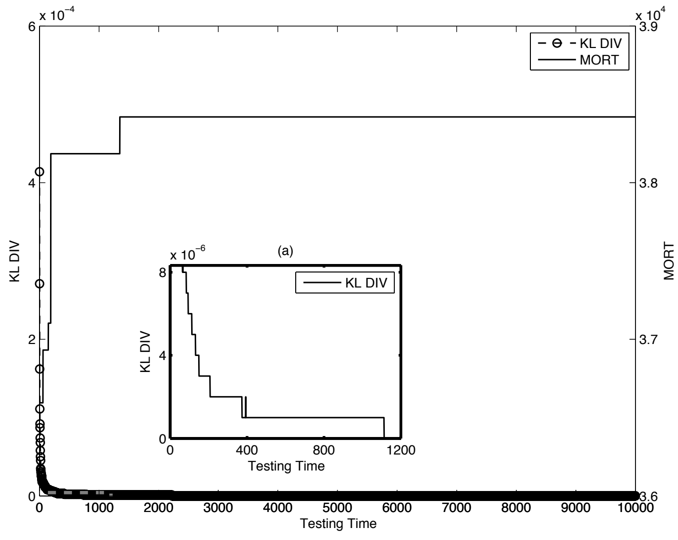

Fig. 1. MORT and KL-DIV versus amount of testing time

The above example shows HWM and KL DIV tests can help for an ALARP decision. However, these tests do not guarantee “no further increase in MORT”. But they claim that such an increase will occur far enough in time. So, there is no significant gain compared to the cost that should be spent on further testing. Figure 2 shows the overall scheme of an algorithm based on HWM and KL DIV tests that is described in the next section.

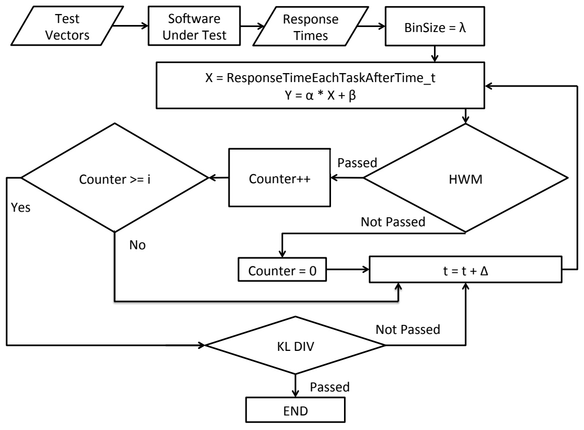

Fig. 2. The Overall Scheme of The Convergence Algorithm

III. Convergence Algorithm

The proposed convergence algorithm is based on HWM and KL DIV tests as described earlier. The tests provide the algorithm the necessary information to decide continuation or termination. Algorithm 1 shows how the convergence algorithm works.

The algorithm inputs test data from the simulator that is run for time *SimulationTime*. The simulator buckets the test data varying between [MinimumObservedResponseTime, MORT] in *bins* every *t* time, generating *SimulationTime*/*t* output files for each task. The binning idea helps to save more memory space compared to output every single response time. The resulted histograms contain the number of occurrence for each observed response time since *BinSize* is set to 1.

While in the algorithm, there is a second binning to prepare data undergo KL DIV test. The data is collected into a number of equally sized of bins depicted by λ in the algorithm, each with size *BinSize* counts the number of response times in range [p, $p + \Delta$]. This avoids any slight difference in data distributions has disproportionate effect on results e.g. a large observed response time that rarely occurs does not significantly affect the test result. *X* and *Y* in the algorithm represent index of each output file that currently undergo the tests.

The idea of comparing histograms of *Probability Distributions (PD) X* and *Y* defined as above is to gradually investigate test data for convergence. So, it avoids further effort as soon as the desired convergence is recognized. The next step is subsequently conducting HWM and KL DIV tests for each task within the generated task set. HWM is a cheap test compared to KL DIV that comes first in analysis. If for *i* successive analysis HWM records no increase ($HWMCounter >= i$), the information undergo KL DIV test. Otherwise, *HWMCounter* is set to zero and later histograms in time are chosen for next round analysis. Once the HWM test is passed, the KL DIV is executed and if this second test result is smaller than a predefined threshold δ, the point at which the current task fulfills tests criteria is kept. The algorithm then processes the next task in the task set.

The tasks in the task set are prioritized based on their deadlines which means the shorter the deadline, the higher priority the task receives. So, when running the simulator for specific duration, the higher priority tasks tend to converge sooner than the lower priority tasks. Due to the late convergence of lower priority tasks, the algorithm keeps track of every individual task's *Stopping Point (SP)* depicted as *Task(CurrentTime, CurrentMORT)* to find out which task is the latest to converge. The algorithm is terminated returning each task MORT at time when the latest task shows convergence.

The convergence algorithm has five parameters λ, α, β, i and δ that affect the analysis results. In the next section, their tuning approach is described.

Algorithm 1: The *Convergence Algorithm*

```
Input: $ResponseTimes$
Output: $AlgorithmStoppingPoint$
foreach Task ∈ {TaskSet} do
    $BinSize \leftarrow (MaxObservedResponseTimeTask - MinObservedResponseTimeTask)/\lambda$;
    $X = 1$;
    $Y = 1$;
    while $Y <= SimulationTime/t$ do
        $Y \leftarrow \alpha * X + \beta$;
        if $(CurrentMORT > OldMORT)$ then
            $HWMCounter \leftarrow 0$;
        end
        else if $(CurrentMORT <= OldMORT)$ then
            $HWMCounter \leftarrow HWMCounter + 1$;
            if $(HWMCounter >= i)$ then
                run KL DIV test;
                if $(KLDIV <= \delta)$ then
                    save current task stopping point coordinates: Task(CurrentTime, CurrentMORT);
                    break;
                end
            end
        end
        else
            $HWMCounter \leftarrow 0$;
        end
        $X \leftarrow X + 1$;
        $OldMORT \leftarrow CurrentMORT$;
    end
end
foreach Point ∈ {TaskSet(CurrentTime, CurrentMORT)} do
    $LatestConvergence \leftarrow Maximum(CurrentTime)$;
    Return Task(Maximum(CurrentTime ), MORT);
end
```

IV. Evaluation

A. Criteria

The convergence algorithm proposes a SP where the ALARP principle is satisfied in terms of sufficient testing for RTS. Two criteria are defined to evaluate the algorithm that are described as follows:

- Closeness of the algorithm SP, *SPMORT* to the *Last MORT (LM)* seen during simulation assuming that virtually infinite test data resources are available. In other words, the simulator is capable of running for ages. So, it is feasible to achieve LM quite close to real worst-case values. Since, the simulator outputs static WCRT, the criterion can be also replaced by closeness of SP to static analysis WCRT. Such a criterion can be expressed as follows:
$$LM - SPMORT <= \epsilon \tag{1}$$
where ϵ is preferred to be as low as possible and can be tuned according to system requirements. To be more precise, this criterion examines if the algorithm results in an acceptable MORT when stops compared to LM. The algorithm fulfills this criterion if there is a reasonable difference between MORT at SP and LM.
- Closeness of SP to a quantified MORT called *ALARP MORT (AM)* in this paper. Ideal is that the algorithm stops later than AM but not far from it. Otherwise, the algorithm fails to fulfill the ALARP principle i.e. by stopping too soon before AM, it misses important information regarding MORT and by stopping too late after AM, it wastes effort without gaining significant new information as we are already within ξ % of the LM at AM. Parameter ξ is tuned based on the problem requirements. In this paper, it is tuned to 5 %. In other words, this criterion avoids the algorithm continues convergence towards LM while ignoring the benefit gained compared to the cost spent. AM is defined as follows:
$$AM <= LM - \xi\% * LM \tag{2}$$
The criterion is as follows:
$$SPMORT - AM <= \epsilon \tag{3}$$
where ϵ should be as low as possible.

These criteria and the algorithm SP are illustrated later in Section IV-E.

B. Method

The analysis is based on a simulation environment. The simulation environment executes a set of randomly generated tasks for a given duration with no system overhead. The tasks are generated with a *Best-Case Execution Time (BCET)*, WCET, a random period and a deadline equal to the period. Tasks's offset and jitter are ignored. The tasks are prioritized based on Deadline Monotonic Priority Ordering (DMPO) i.e. for each task the shortest the deadline, the highest priority it takes. Each task is assigned a random execution time in a range [BCET, WCET] as the simulation starts. The scheduler monitors each task's status: *delayed*, *in run queue* or *executed* and performs preemptive scheduling based on tasks' fixed priorities. The simulator is also capable of generating each tasks's WCRT by static analysis.

C. Experimental Setup

The analysis is based on observed response times generated by the simulator. The simulator initial setup is as follows:

- Choice of simulator set up i.e. *SimulationDuration* that indicates how long the simulator has to run such that MORT for each task becomes close to realistic worst-case values. For this, MORT achieved by the simulator can be compared to WCRT by static analysis. However, practically, it is not possible to run the simulator for ages due to time issue compared to MORT achieved. For instance,

TABLE I
MORT vs. Static Analysis WCRT

Task	MORT (10^9)	MORT (10^{12})	MORT (10^{14})	MORT (10^{15})	WCRT (static analysis)
1	3493	3494	3494	3494	3494
2	4897	4906	4907	4907	4908
3	7913	8030	8039	8040	8043
4	12145	12397	12501	12527	12534
5	12976	13892	13995	14082	14102
6	13565	14433	14743	14827	14941
7	14556	15699	16147	16215	16376
8	15433	16744	17189	17294	17619
9	16582	17811	18295	18545	18976
10	18391	19725	23377	25065	26257
11	24289	25748	26742	27227	29031
12	24683	26450	27345	27991	29832
13	26392	31894	33980	34253	37118
14	29033	32884	33625	34448	37544
15	29479	33366	34061	34885	38035
16	29623	33559	34284	35129	38302
17	32235	35867	36890	37669	55581
18	32581	36227	37015	38255	56183
19	32419	37357	39333	39839	58674
20	32904	37990	41685	47785	59621

Algorithm 2: The *Task Set Generation Algorithm*

```
Input: $NumberOfTasks, MinPeriod, MaxPeriod, PeriodStep, MaxBCET, BCETStep, MaxWCET, WCETStep$
Output: $TaskSetCharacteristics$
$NumberOfTasks = 20$;
foreach $Task \in \{TaskSet\}$ do
    $TaskPeriod \leftarrow Rand(MinPeriod, MaxPeriod, PeriodStep)$;
    $TaskDeadline \leftarrow TaskPeriod$;
    $TaskBCET \leftarrow Rand(1, MaxBCET, BCETStep)$;
    $TaskWCET \leftarrow Rand(BCET, MaxWCET, WCETStep)$;
    $TaskOffset = 0$;
    $TaskJitter = 0$;
    $TaskPriority \leftarrow DMST(TaskSet)$;
end
```

table I shows MORT versus WCRT static results for 20 tasks with different simulation durations. As it can be seen in this table, column two corresponds to *Simulation-Duration* equal to 10^9 that takes a couple of minutes to be completed. However, comparing these results with longer simulation durations indicates that significant improved MORT can be gained by spending a feasible simulation time. For instance, the fourth column shows MORT when *SimulationDuration* is equal to 10^{14} that takes a couple of hours to be completed. The fifth column presents the results when the simulation is run 10 times longer which approximately takes one week to completion. However, specifically for lower priority tasks, we have not gained highly improved MORT in terms of closeness to static analysis results (shown in sixth column) compared to the cost we spent (time in this case). The reason for this difference is that due to the way the task set is generated (i.e. the ranges for [BCET,WCET] and task periods span five orders of magnitude), then the worst-case situation is extremely unlikely to be seen in practice. Consequently, the *SimulationDuration* is set to the longest practical value according to our experiments (10^{14}). The simulator outputs 10000 files of response times for each task with period *SimulationDuration/10000*.

- Task set generation is based on the following characteristics of type integer for each task:
  - *WCET* the longest execution time,
  - *BCET* the shortest execution time,
  - *Period* time interval in which a job of a task is released,
  - *Deadline* time by which a job of a task has to be completed,
  - *Offset* determines the precedence of tasks' execution for tasks with the same period,
  - *Jitter* time deviation from predefined periodic task's release time

  and takes place according to the Algorithm 2. *Rand* function in the algorithm is initialized by a random seed and returns a random value in range [MinValue, MaxValue] with coefficient *ValueStep* in which *Value* can stand for period, BCET or WCET. *DMST* function returns each task priority based on DMPO.

  All task sets have a utilisation in the range 80% to 100% and the ones that are schedulable with static analysis are only used in experiments. The task set also has one transaction of 3 tasks.

D. Tuning of Parameters

As per the convergence algorithm, there are five parameters influencing the analysis results. They are experimentally tuned as follows:

- λ - For tuning λ, three random variables are selected that lead to relatively small, middle and large BinSize. Consequently, λ is tuned to the value that causes meaningful KL DIV results. As an example, Figure 3 (a) shows the zoomed KL DIV test result of a task for $\lambda = 63$ while (b) shows KL DIV of the same task for $\lambda = 250$. Looking at KL DIV test results, (a) with a big *BinSize* does not lead to any helpful test results while KL DIV in (b) has step change convergence that complies MORT gradual increase over time and causes a better decision for stopping the analysis. The quick convergence in (a) due to the large bins misses differences in PD compared to the smaller bin size in (b). Based on the achieved results, λ is tuned to 250. However, optimizing the parameters may lead to better results. Future work will look at how to optimise the parameters.
- α, β - For tuning these two parameters we try two alternatives: $\alpha = 2$, $\beta = 0$ and $\alpha = 1$ and $\beta = p + q - 1$ where p and q are incremented by 1 in each round of analysis. The results of the latter tuning lead into shorter number of analysis with sharp increase in KL DIV compared to the former. This does not allow observing slight KL DIV changes over time which is not suitable for making a stopping decision. Therefore, we tuned these two parameters to the former values. However, there might be more promising values that could be achieved by optimization.
- i - Tuning i to larger values than 100 e.g. 200 and observing the algorithm behaviour shows no improved results but wasting testing effort as KL DIV has already converged to the threshold. So, we tuned i to 100 in all experiments.

- δ - We observe lower priority tasks with complex timing behaviour need a very small threshold (δ = 0.000003). Tuning δ to bigger values e.g. 0.0005 causes the algorithm stops too soon compared to MORT for tasks with complex timing behaviour.

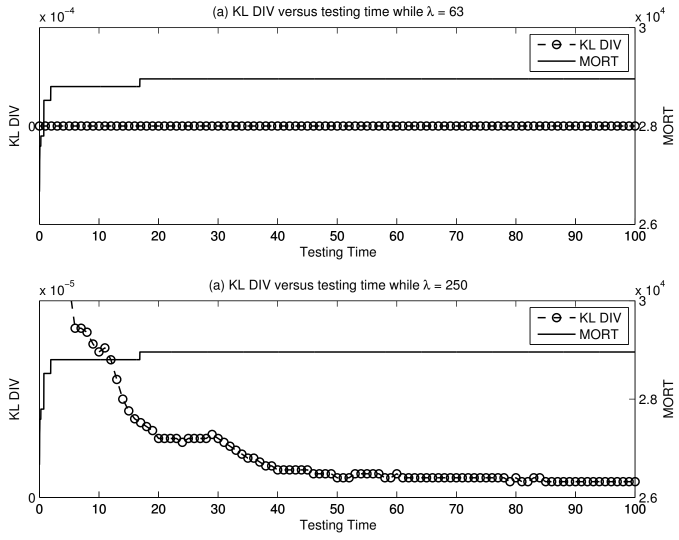

Fig. 3. MORT and KL-DIV versus amount of testing time

E. Results

Results are based on the analysis of the simulator's outputs. We run 20 experiments. Then, analyze the outputs to examine the convergence algorithm performance. The convergence algorithm is scripted in MATLAB.

Figures 4 and 5 present the convergence algorithm performance for two arbitrary experiments. In both figures, (a) and (b) show the algorithm SP versus LM, AM and HWM test that are denoted by SP, LM, AM and HWM respectively. The horizontal axis shows the number of tests over time while the vertical axis presents MORT during the simulation. For simplicity, the analysis is presented as soon as LM is encountered. In both figures, the algorithm behaviour is shown for (a) a high priority task and (b) a low priority task. In Figure 4 (a), SP occurs at point (432, 19697) where the first coordinates stands for testing time and the second one shows the corresponding MORT. MORT at SP is within 101 time units to LM (7318, 19798) and 506 time units from AM (2, 19191).

We intentionally select a rare case that is shown in (b) where the algorithm stops earlier compared to AM. The task has a low priority in the task set with complex RT behaviour such that RT gets significant increased late in testing time. This causes AM (3112, 46491) occurs late close to LM (3874, 48204). Such an AM does not fulfill ALARP principle due to significant testing time. So, the algorithm SP seems reasonable compared to the late AM. Based on the ALARP principle, MORT at SP (432, 44210) is acceptable in terms of effort compared to LM (3874, 48204).

In this figure, (c) and (d) present histograms of the following data achieved by the convergence algorithm:

- *Algorithm Achievement (AA)* defined as follows:

$$AA = (SPMORT/LM) * N \tag{4}$$

for which the larger the value, the better indication of algorithm success. Since it is desired that the algorithm stops relatively close to LM that results in larger value of fraction above.

- *Algorithm Effort (AE)* presented as follows:

$$AE = (TestingTimeatSP/TestingTimeatLM) * N \tag{5}$$

for which the smaller the value, the more successful the algorithm as the algorithm is expected to stop within an acceptable distance from the LM while considering the spent effort at the same time (ALARP).

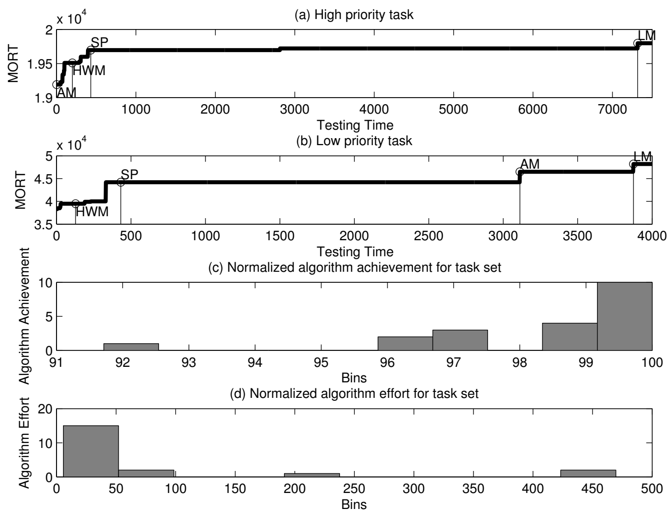

Fig. 4. The Convergence algorithm performance

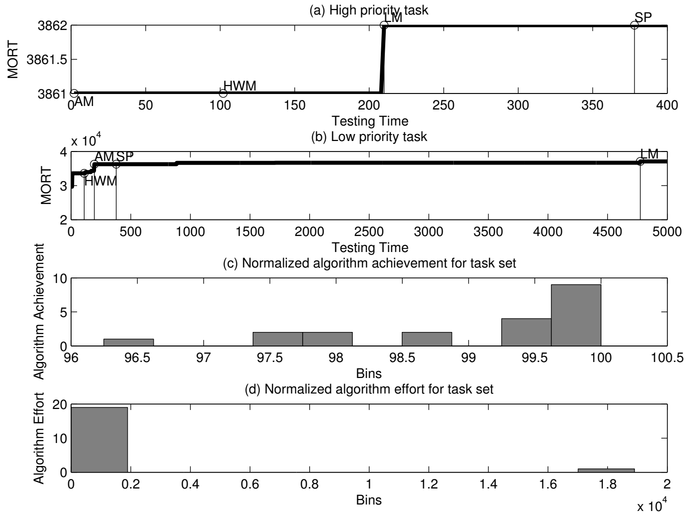

Fig. 5. The Convergence algorithm performance

The histograms are normalized by N = 100. In this experiment, mean value for normalized AA and AE is 98.42 and 76.36 respectively. It can be seen in (c) that AA mean value in the range of bins [91, 100] is relatively large indicating that most of the tasks' SP occurs within an acceptable range to MORT according to Equation 1 while (d) shows average testing time spent to reach SP (AE mean = 76.36) for the task set is small compared to the bin ranges [0, 500] according to Equation 2. From (c) and (d) in Figure 4, it can be concluded that the algorithm performs as desired in this experiment i.e. a fair trade-off between achieved MORT versus testing time (the ALARP principle).

Figure 5 shows the same graphs as Figure 4 for another experiment. In (a), the highest priority task timing behaviour is depicted for which the algorithm stops even passing LM at point (378, 3862). This is due to the late convergence of a lower priority task within the task set. AM and LM occur at points (2, 3861) and (210, 3862) respectively. The (b) graph presents SP for a lower priority task with AM (194, 36259), SP (378, 36259) and LM (4774, 37087). HWM occurs at point (110, 33602) that shows how the KL DIV test can improve the algorithm performance after HWM test is passed. The histogram presented in (c) has AA mean value equal to 99.95 which is close to the upper bound in range [96, 100]. In (d), AE mean is equal to 1011 which is relatively small in range [0, 20000]. The overall results show the algorithm fulfills the ALARP principle i.e. acceptable MORT within a reasonable testing time.

V. Conclusions

This paper has explained the importance of quantified decisions of when sufficient testing has been performed according to the principle ALARP. A novel convergence algorithm has been proposed that decides when sufficient testing has been performed. An evaluation is performed of how well the algorithm decides when the testing is sufficient compared against a ground truth. The work shows that the convergence algorithm performs well. That is, testing is stopped after the point at which a human with clairvoyance would stop the testing, ensuring a safe WCRT estimate is obtained, but the amount of unneeded testing is reasonable. Future work will perform more trials of the algorithm, including on real software, and investigate how the algorithms can be tuned for more robust performance, i.e. so the parameters hold across many task sets.

Acknowledgement

We acknowledge the Swedish Foundation for Strategic Research (SSF) SYNOPSIS Project for supporting this work.

References

[1] "Health and safety at work act 1974," The National Archives, Tech. Rep., 1974. [Online]. Available: http://www.legislation.gov.uk/ukpga/1974/37/contents

[2] T. Aven and E. Abrahamsen, "On the use of cost-benefit analysis in alarp processes," *International Journal of Performability Engineering*, vol. 3, no. 3, pp. pp. 345 – 353, July 2006.

[3] R. Weaver, J. McDermid, and T. Kelly, "Software safety arguments: Towards a systematic categorisation of evidence," in *Proceedings of the 20th International System Safety Conference, System Safety Society*, 2002.

[4] I. Bate and A. Burns, "An integrated approach to scheduling in safety-critical embedded control systems," *Real-Time Systems Journal*, vol. 25, no. 1, pp. 5–37, Jul 2003.

[5] P. Graydon and I. Bate, "Realistic safety cases for the timing of systems," *The Computer Journal*, February. [Online]. Available: http://www.mrtc.mdh.se/index.php?choice=publications&id=3270

[6] Y. Lu, T. Nolte, L. Cucu-Grosjean, and I. Bate, "Rapidrt: A tool for statistical response-time analysis of complex industrial real-time embedded systems," in *Real-Time SystemS @ Work, the Open Demo Session of Real-Time Techniques and Technologies of the 32nd IEEE Real-Time Systems Symposium (RTSS'11)*, November 2011.

[7] L. Cucu-Grosjean, L. Santinelli, M. Houston, C. Lo, T. Vardanega, L. Kosmidis, J. Abella, E. Mezzetti, E. Quiñones, and F. J. Cazorla, "Measurement-based probabilistic timing analysis for multi-path programs." in *Proceedings of the Euromicro Conference on Real-Time Systems, 2012*, pp. 91–101.

[8] Z. Stephenson, J. Abella, and T. Vardanega, "Supporting Industrial Use of Probabilistic Timing Analysis with Explicit Argumentation," in *Proceedings of the IEEE 11th International Conference on Industrial Informatics*.

[9] E. Gumbel, *Statistics of Extremes*, ser. Dover books on mathematics. Dover Publications, 2004.

[10] S. Edgar and A. Burns, "Statistical analysis of wcet for scheduling," in *Proceedings of the 22th IEEE Real-Time Systems Symposium*, 2001.

[11] G. Bernat, A. Colin, and S. M. Pettersreal-time, "Wcet analysis of probabilistic hard real-time systems," in *In Proceedings of the 23rd Real-Time Systems Symposium RTSS 2002*, 2002, pp. 279–288.

[12] A. Burns and A. Wellings, *Real-time Systems and Programming Languages: Ada 95, Real-time Java, and Real-time POSIX*, ser. International computer science series. Addison-Wesley, 2001.

[13] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Muller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström, "The worst-case execution-time problem–overview of methods and survey of tools," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 7, no. 3, pp. 1—53, April 2008.

[14] I. Bate and A. Burns, "An integrated approach to scheduling in safety-critical embedded control systems," *Real-Time Syst.*, vol. 25, no. 1, pp. 5–37, Jul. 2003.

[15] J. Kraft, Y. Lu, C. Norström, and A. Wall, "A metaheuristic approach for best effort timing analysis targeting complex legacy real-time systems," in *the 15th IEEE Real-Time and Embedded Technology and Applications Symposium*, 2008, pp. 258–269.

[16] A. Burns, G. Bernat, and I. Broster, "A probabilistic framework for schedulability analysis," in *Proceedings of the Third International Conference on Embedded Software (EMSOFT 2003*, 2003, pp. 1–15.

[17] Y. Lu, T. Nolte, I. Bate, and L. Cucu-Grosjean, "A statistical response-time analysis of complex real-time embedded systems by using timing traces," in *Proceedings of the 6th IEEE International Symposium on Industrial Embedded Systems (SIES'11), Work-in-Progress (WiP) session*, 2011.

[18] C.-G. Lee, J. Hahn, Y.-M. Seo, S. L. Min, R. Ha, S. Hong, C. Y. Park, M. Lee, and C. S. Kim, "Analysis of cache-related preemption delay in fixed-priority preemptive scheduling," *IEEE Transactions on Computers*, vol. 47, no. 6, pp. 700–713, 1998.

[19] S. Chattopadhyay, A. Roychoudhury, and T. Mitra, "Modeling shared cache and bus in multi-cores for timing analysis," in *Proceedings of the 13th International Workshop on Software; Compilers for Embedded Systems*, 2010.

[20] H. Kopetz, *Real-Time Systems: Design Principles for Distributed Embedded Applications*, 2nd ed. Kluwer Academic Publishers, 1997.

[21] D. Griffin and A. Burns, "Realism in statistical analysis of worst case execution times," in *Proceedings of the 10th International Workshop on Worst-Case Execution Time Analysis*, 2010, pp. 49–57.

[22] S. Kotz and S. Nadarajah, *Extreme Value Distributions: Theory and Applications*. Imperial College Press, 2000.

[23] W. Feller, *An Introduction to Probability Theory and Its Applications*. Wiley, January 1968, vol. 1.

[24] J. Diebolt, M. Garrido, and S. Girard, "A Goodness-of-fit Test for the Distribution Tail," in *Topics in Extreme Values*, 2007, pp. 95–109, chapitre 5.

[25] S. Kullback and R. A. Leibler, "On information and sufficiency," *The Annals of Mathematical Statistics (AMS)*, vol. 22, no. 1, pp. 79–86, 1951.

[26] S. Kullback, *Information Theory and Statistics*. Wiley, 1959.