WATERS

1^{st} International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS 2010)

July 6th, 2010, Brussels, Belgium

ECRTS
July 6-9, 2010
Brussels, Belgium
2010

In conjunction with the 22^{nd} Euromicro Conference on Real-Time Systems (ECRTS10)

Table of Contents

Message from the Program Chairs

Research in real-time systems has gone very far from the initial seminal papers back in the 70s. Many algorithms, design methodologies, techniques and tools have been proposed, spanning several application areas, from Real-Time Operating Systems to distributed systems, from safety critical to soft real-time systems. However, unlike other research areas (e.g., networking) there are no widely recognized reference tools or methodologies for comparing different research works in the area.

In fact, the comparison among results achieved by different research groups becomes non-trivial or impossible due to the lack of common tools or methodologies by means of which the comparison is done. For example, different authors use different algorithms for generating random task sets, different application traces when simulating dynamic real-time systems, different simulation engines when simulating scheduling algorithms. Therefore, research in the field of real-time and embedded systems would greatly benefit from the availability of well-engineered, possibly open tools, simulation frameworks and data sets which may constitute a common metrics for evaluating simulation or experimental results in the area. Also, it would be nice to have a possibly wide set of reusable data sets or behavioural models coming from realistic industrial use-cases over which to evaluate the performance of novel algorithms. Availability of such items would increase the possibility to compare novel techniques in dealing with problems already tackled by others from the multifaceted viewpoints of effectiveness, overhead, performance, applicability, etc.

The ambitious goal of the International Workshop on Anaysis Tools and Methodologies for Embedded and Real-time Systems (WATERS) is to start creating a common ground and a community to collect methodologies, software tools, best practices, data sets, application models, benchmarks and any other way to improve comparability of results in the current practice of research in real-time and embedded systems. People from industry are welcome to contribute with realistic data or methods coming from their own experience.

The WATERS Workshop collects original contributions on methods and tools for real-time and embedded systems analysis, simulation, modelling and benchmarking. In particular, areas of interest include, but are not limited to:

- Simulation of real-time, distributed and embedded systems
- Tools and methodologies for real-time analysis
- Instrumentation of Operating Systems
- Tracing methods and overhead analysis

- Power consumption models and experimental data for real-time power-aware systems
- Realistic case studies and reusable data sets
- Comparative evaluation of existing algorithms

We would like to thank the Euromicro organization for having allowed us to organize this event, and particularly Gerhard Fohler for his prompt and ready support. We would like to thank all the authors for having submitted their work to the workshop for selection, the Program Committee members for their effort in reviewing the papers, the presenters for ensuring interesting sessions, and the attendees for participating into this event. We hope that interesting ideas and discussions will come out of the presentations, demos and the questions that will alternate along the day. We hope you will find this day interesting and enjoyable.

The WATERS 2010 Chairs
Giuseppe Lipari and Tommaso Cucinotta

Real-Time Systems Laboratory
Scuola Superiore Sant'Anna, Pisa (Italy)
{`g.lipari, t.cucinotta`} `@ sssup.it`

Program Committee

Techniques For The Synthesis Of Multiprocessor Tasksets

Paul Emberson
Rapita Systems Ltd
IT Centre, York Science Park, York, YO10 5DG
paule@rapitasystems.com

Roger Stafford
CA, USA

Robert I. Davis
Department of Computer Science
University of York, York, YO10 5DD
robdavis@cs.york.ac.uk

***Abstract*—The selection of task attributes for empirical evaluations of multiprocessor scheduling algorithms and associated schedulability analyses can greatly affect the results of experiments. Taskset generation algorithms should meet three requirements: efficiency, parameter independence, and lack of bias. Satisfying these requirements enables tasksets to be generated in a moderate amount of time, allows effects of specific parameters to be explored without the problem of confounding variables, and ensures fairness in comparisons between different schedulability analysis techniques. For the uniprocessor case, they are met by the UUniFast algorithm but for multiprocessor systems, where the total desired utilisation is greater than one, UUniFast can produce invalid tasksets. This paper outlines an algorithm, Randfixedsum, for the underlying mathematical problem of efficiently generating uniformly distributed random points whose components have constant sum. This algorithm has been available via a MatLab forum for a number of years; however, this is the first time it has been formally published. This algorithm has direct application to multiprocessor taskset generation. The importance of period generation to experimental evaluation of schedulability tests is also covered.**

I. Introduction

To address demands for increasing processor performance, silicon vendors no longer concentrate on increasing processor clock speeds, as this approach is leading to problems with high power consumption and excessive heat dissipation. Instead, there is now an increasing trend towards using multiprocessor platforms for high-end real-time applications. As a result, multiprocessor task allocation and scheduling has become an important and popular area of research.

While optimal algorithms and exact schedulability tests are known for uniprocessor scheduling, multiprocessor scheduling is intrinsically a much more difficult problem due to the simple fact that a task can only use one processor at a time, even when several are free. As a result, no efficient algorithms are known that can optimally schedule general sporadic tasksets (without restrictions on deadlines). Much of the research into multiprocessor scheduling has therefore involved the analysis of heuristic scheduling policies, and the development of sufficient schedulability tests.

A number of different performance metrics can be used to assess the effectiveness of multiprocessor scheduling algorithms and their analyses. These include: optimality, comparability (or dominance) [1], utilisation bounds [2], resource augmentation or speedup factors [3], and empirical measures such as the number of tasksets that are deemed schedulable.

The research in this paper is motivated by empirical approaches to evaluating scheduling algorithm and schedulability test performance. A systematic and scientific study of the effectiveness of different scheduling algorithms and analyses requires a method of synthesising tasksets to which the scheduling algorithms and tests can be applied. We can identify three key requirements of this *taskset generation problem*: efficiency, independence, and bias.

1) *Efficient* — in order to achieve statistically significant sample sizes, large numbers of tasksets need to be generated for each taskset parameter setting (or data point) examined in experiments.
2) *Independent* — it should be possible to vary each property of the taskset independently. For example, experiments might examine the dependency of schedulability test effectiveness on the number of tasks, on taskset utilisation or on the range of task periods. The parameter of interest must be varied independent of other parameters which are held constant.
3) *Unbiased* — the distribution of tasksets generated should be equivalent to selecting tasksets at random from the set of all possible tasksets, and then discarding those that do not match the desired parameter setting.

We assume the sporadic task model commonly used in real-time systems research. A sporadic taskset comprises n tasks with the following attributes: period or minimum inter-arrival time T_i, worst-case execution time C_i and deadline D_i. The utilisation of a task is defined as $U_i = C_i/T_i$. Two important taskset parameters used for understanding the behaviour of scheduling algorithms and their analyses are the taskset cardinality n and the total taskset utilisation u. Hence we are interested in taskset generation algorithms that select utilisation values U_i so that:

$$\sum_{i=1}^{n} U_i = u \tag{1}$$

for n tasks where the target total utilisation is u. Once periods have also been generated, worst-case execution times can then

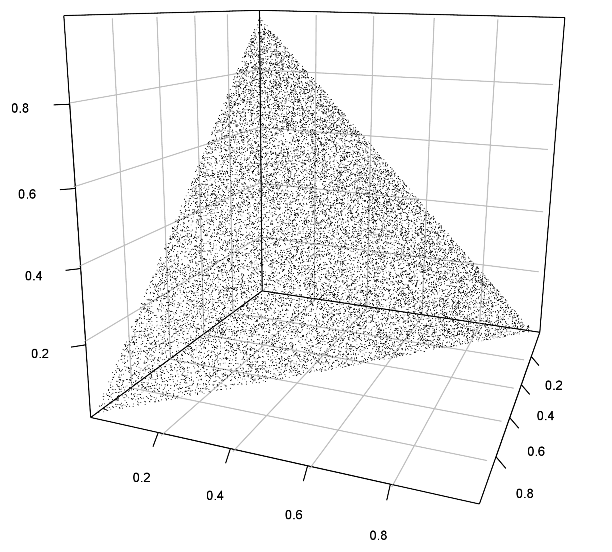


Fig. 1. $2 \cdot 10^4$ tasksets generated by UUniFast with total utilisation 0.98

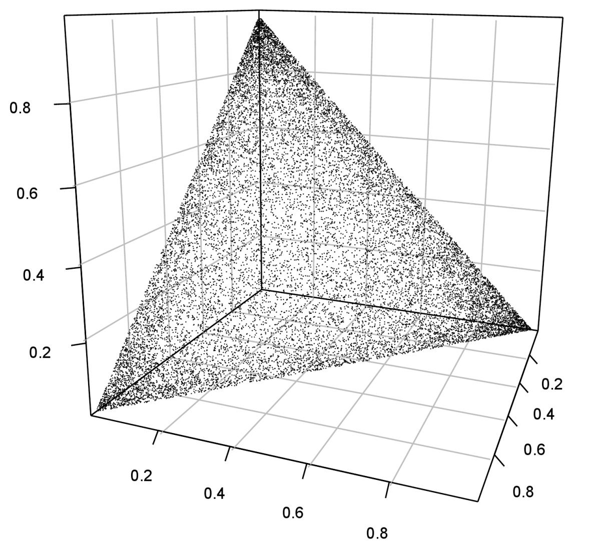


Fig. 2. Subset of 10^5 tasksets which can be scheduled by rate monotonic fixed priority scheduling (approx $1.8 \cdot 10^4$ tasksets)

be set with the formula:

$$C_i = U_i T_i \tag{2}$$

Task deadlines must also be selected. These can be set equal to T_i or randomly generated based on a proportion of the task's period or execution time. This paper focuses mainly on selecting utilisation values though a method of task period generation is given in section III.

II. Related Work

A. Uniprocessor Taskset Generation

In 2005, Bini and Buttazzo [4] created an algorithm called UUniFast that efficiently generates task utilisation values for tasksets with a chosen number of tasks and total utilisation. The distribution of utilisation values in tasksets generated by UUniFast are equivalent to uniformly sampling each task utilisation value and then only keeping those tasksets with the correct total utilisation. A taskset containing n tasks can be plotted in an n dimensional space where the utilisation of each task gives the distance from the origin in each dimension. If this is done for a set of tasksets all having the same total utilisation, then the tasksets will lie in an $n-1$ dimensional plane. Tasksets generated by the UUniFast algorithm will be evenly separated in this plane. Figure 1 shows 20000 tasksets containing 3 tasks generated by UUniFast all having a total utilisation of 0.98.

Bini and Buttazzo considered experiments which evaluated how many tasksets could be scheduled using rate monotonic fixed priority scheduling versus other scheduling policies. They noted that, if periods are uniformly sampled, tasksets are more often schedulable when the difference between the greatest and least task utilisation is large. This phenomenon is shown by figure 2. The plot shows the subset of 10^5 tasksets, again generated by UUniFast with a utilisation of 0.98, deemed schedulable using rate monotonic fixed priority scheduling with periods uniformly sampled between 10 and 10^4. 17953 tasksets are contained within this subset, i.e. a similar number of tasksets as shown in figure 1. The points in figure 2 appear more densely packed towards the edges of the plane whereas those in figure 1 are evenly distributed.

The motivation for Bini and Buttazzo's work was that previous evaluations of scheduling policies, such as by Lehoczky et al. [5], had biased results by concentrating on the area in the centre of the plane shown in figure 2 where fewer tasksets can be scheduled by rate monotonic fixed priority scheduling.

The UUniFast algorithm is efficient, allows variable independence, and generates unbiased utilisation values. UUniFast has been widely used by researchers interested in investigating the performance of scheduling algorithms and schedulability tests for single processors [6], [7].

B. Multiprocessor Taskset Generation

In the multiprocessor domain, the UUniFast algorithm has not been widely used. Researchers recognised that the algorithm cannot generate tasksets with total utilisation $u > 1$ without the possibility that some tasks will have individual utilisations that are invalid (i.e. > 1). Instead, many researchers [8], [9], [10], [11] have used an approach to taskset generation based on randomly generating an initial taskset of cardinality $|\mathcal{P}| + 1$ for the set of processors $\mathcal{P}$ and then repeatedly adding tasks to it until the total utilisation exceeds the available processing resource. This approach has the disadvantage that it confounds two variables, utilisation and taskset cardinality, and does not necessarily result in an unbiased distribution of utilisation values.

Recently, Davis and Burns [12] observed that UUniFast can be used in the multiprocessor domain, at least for some values of n and u, provided that tasksets containing invalid tasks are simply discarded. We give more details of this modified UUniFast algorithm, referred to as UUniFast-Discard in section IV-B. While UUniFast-Discard addresses a proportion of the parameter space of n and u, there are values of n and u where this approach becomes infeasible, due to the very high ratio of invalid to valid tasksets produced.

This paper addresses the problem of generating tasksets for multiprocessor systems. Stafford's *Randfixedsum* algorithm [13] is used to generate unbiased sets of utilisation values for any values of n and u.

The remainder of the paper is broken into two main sections. Section III discusses the associated issue of task period selection. Section IV explains why existing algorithms for generating tasksets with total utilisation greater than 1 are inadequate and suggests the use of the Randfixedsum algorithm. Section V concludes with a summary of the main contributions of the paper.

III. Task Period Selection

In this section, we discuss task period selection. In commercial real-time systems, it is common for systems to have tasks operating in different time bands [14] (e.g. 1ms – 10ms, 10ms – 100ms, 100ms – 1s). For example, a temperature sensor will likely sample at a lower rate than a rotation speed sensor [15].

Davis et al. [6] showed that schedulability test efficiency can be heavily dependent on the number of order of magnitude ranges of task periods (effectively the ratio between the smallest and largest task period), and that bias can result if studies do not fully explore appropriate distributions of task periods. For example, choosing task periods at random according to a uniform distribution in the range $[1, 10^6]$ results in 99% of tasks having periods greater than 10^4, thus the effective ratio of maximum to minimum task period is far less than might be expected (closer to 10^2 than 10^6 for small tasksets).

To avoid these problems, a log-uniform distribution of task periods can be used, with tasksets generated for different ratios of the minimum (T_{min}) to the maximum (T_{max}) task period. The parameter T_g defines the granularity of the periods chosen (which are all multiples of T_g).

$$r_i \sim U(\log T_{min}, \log(T_{max} + T_g)) \tag{3}$$

$$T_i = \left\lfloor \frac{\exp(r_i)}{T_g} \right\rfloor T_g \tag{4}$$

The uniform random values r_i produced are assumed to lie in the range $[\log T_{min}, \log(T_{max} + T_g))$. T_{min} and T_{max} should be chosen as multiples of T_g.

Note that when applying equation (2) the worst-case execution time is usually rounded to the nearest integer which will affect the distribution of actual utilisations in the generated taskset. Changing the unit of time to use larger numeric values will decrease this loss of accuracy.

The effects of different period sampling algorithms were examined with some simple experiments. In each case 1000 tasksets were generated and tested using exact schedulability analysis for fixed priority pre-emptive scheduling on a uniprocessor [16]. Periods were sampled from either a uniform or log-uniform distribution within a certain range. The correct total utilisation for a chosen number of tasks was achieved

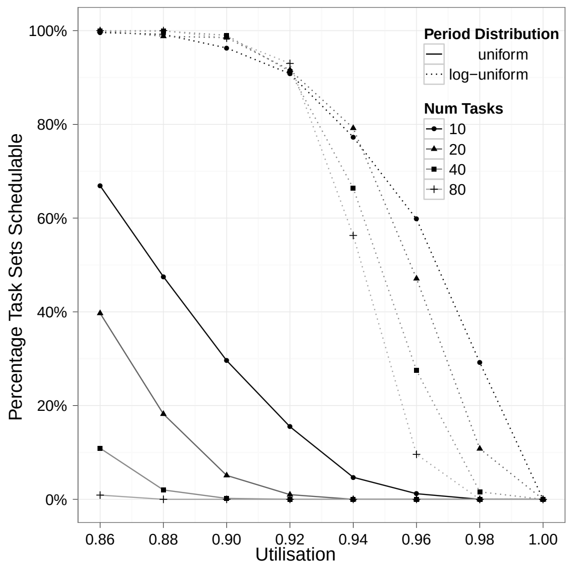


Fig. 3. Comparison of taskset schedulability for different size uniprocessor tasksets generated with uniform and log-uniform period distributions in the range $[10, 10000]$.

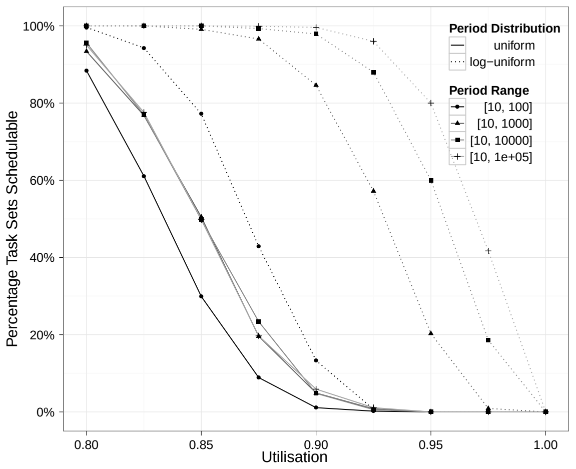


Fig. 4. The effect of changing the range of periods on taskset schedulability for tasksets of size 20 generated with uniform and log-uniform period distributions.

with the UUniFast algorithm. Taskset deadlines were set equal to their periods and priorities assigned according to rate monotonic priority ordering.

Figure 3 shows the proportion of schedulable tasksets for varying taskset cardinality and total utilisation. The period range was set to $[10, 1000]$ for all experiments. The plot shows that many more tasksets are schedulable when taskset periods are sampled from a log-uniform distribution for all utilisation levels up to 0.98. The difference in the number of schedulable tasksets is also much smaller at lower utilisation values over the different taskset sizes when using log-uniform sampling.

Lehoczky [5] calculated that tasksets with a greater range of periods would be easier to schedule using exact rate monotonic analysis. Lehoczky assumed periods were sampled from a uniform distribution. This is supported by the results shown in figure 4. The graph shows the proportion of schedulable tasksets for different period ranges and total utilisations. All tasksets were of size 20. These results show the phenomenon described by Davis [6] that, when a uniform distribution of periods is used, the number of schedulable tasksets does not continue to increase for large period ranges because nearly all period values will be of the same magnitude. The period range has a much larger effect on schedulability of periods sampled from a log-uniform distribution. In fact, for uniform period sampling, there is no significant increase in schedulability as the range widens to more than a factor of 100 and many results overlap on the graph. Even a range whose maximum is only 10 times greater than its minimum produces more schedulable tasksets with a log-uniform period distribution than ranges over 4 orders of magnitude when uniform sampling is used.

IV. Task Worst Case Execution Time Generation

Rather than generate worst case execution time (WCET) values directly, it is more common to generate task utilisation values then calculate WCET values from equation (2). This is done since the total taskset utilisation is an often used covariate in experiments with schedulability tests. This total taskset utilisation value is written as u in this paper. The other common covariate is taskset cardinality and this should be possible to control independently from taskset utilisation.

A. *UUniFast*

Motivated by the need for an unbiased distribution of tasksets, Bini and Buttazzo [4] decided that task utilisation values should be sampled from a uniform distribution but with the constraint that they summed to a constant desired total taskset utilisation. An algorithm for doing this is to randomly select utilisation values $x_1, \ldots, x_{n-1} \sim U(0,1)$ and then set $x_n = 1 - \sum_{i=1}^{n-1} x_i$. However, if the sum term is greater than 1, the set must be discarded and the operation repeated. If successful, utilisation values are set according to $U_i = ux_i$. Bini and Buttazzo call this algorithm UUniform and explain that it is infeasible in practice since the probability that the sum of the first $n-1$ values is less than u is $1/(n-1)!$ [4].

The UUniFast algorithm [4] is an efficient equivalent of the above algorithm. The principle of the algorithm is to first sample a value which represents the sum of $n-1$ task utilisation values and then set a task utilisation value to the difference between the required total and this sampled value. This is then repeated for each task with the sampled value in the previous iteration acting as the required total.

The probability density function for the sum of m independent random variables uniformly selected from $[0,1]$ is

$$UniSumPdf(x;m) = \frac{1}{(m-1)!} \sum_{k=0}^{\lfloor x \rfloor} (-1)^k \binom{m}{k} (x-k)^{m-1} \tag{5}$$

We refer to this distribution as the UniSum distribution. It is adapted from Hall's derivation for the density of the mean of m independent uniform random variables [17]. It is a piecewise function where each region $[a, a+1]$ for $a = 0, \ldots, m-1$ is defined by a different polynomial of degree $m-1$. Therefore, if we wish to sample a value which represents the sum of $n-1$ utilisation values as required for UUniFast, m is set to $n-1$. The graph of this probability density function is shown in figure 5. The domain which must be sampled from for UUniFast is $[\max(u-1, 0), u]$. For uniprocessor tasksets, $u \leq 1$. The relevant area of the graph is highlighted in figure 5. The cumulative distribution function in this region is proportional to $x^m = x^{n-1}$ and is easily invertible. UUniFast makes use of this fact to perform inverse transform sampling in order to obtain values for the sum of $n-1$ values. The UUniFast algorithm is given below.

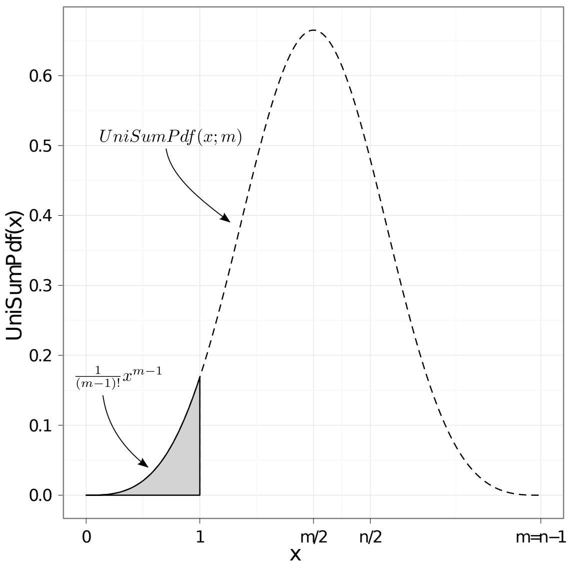


Fig. 5. UniSum probability density function which could be used to sample the sum of $n-1$ values.

```
Let $r_1, \ldots, r_{n-1} \sim U(0,1)$
$s_n = u$
$s_{i-1} = s_i * {r_{i-1}}^{1/(i-1)}$ for $i = n, \ldots, 2$ and $s_0 = 0$.
$u_i = s_i - s_{i-1}$
```

There are a few points of note regarding extending UUniFast for total taskset utilisation values $u > 1$. The distribution given by equation (5) is symmetrical about $(n-1)/2$. If an algorithm can sample values for $0 \leq u \leq n/2$ then sampling values for a total utilisation $u' > n/2$ can be obtained by sampling values with $u = n - u'$ and then using $u'_i = 1 - u_i$ for each task utilisation value.

The complex piecewise nature of the UniSum distribution makes it difficult to sample from in the general case. The sum of n independent random variables will approach a normal distribution but the accuracy of the approximation is heavily dependent on the number of tasks and region of the distribution being sampled from. Saddlepoint approximations [18], [19] are more accurate. However, in either case, sampling from a

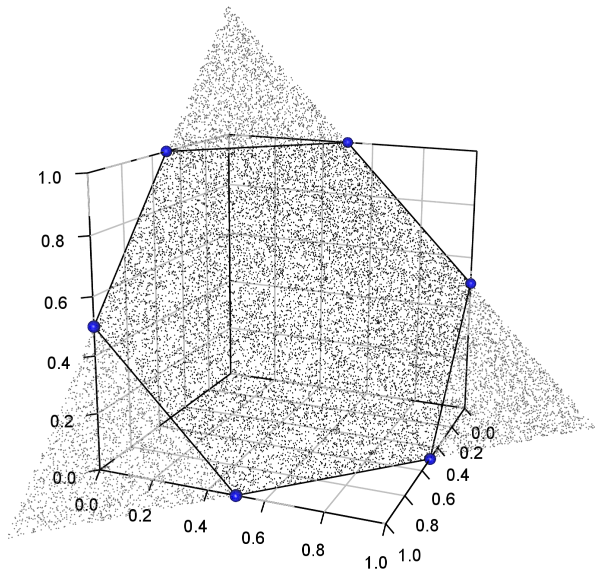


Fig. 6. Tasksets generated with UUniFast-Discard for $n = 3, u = 1.5$

truncated section of the distribution is difficult to do efficiently since it usually requires rejecting a number of samples as well as calculating the distribution itself.

B. UUniFast-Discard

UUniFast-Discard is a simple extension to UUniFast suggested by Davis and Burns [12]. This algorithm applies UUniFast unchanged for values of $u > 1$ and then discards any tasksets which contain an individual task utilisation greater than 1. The issue with this algorithm is that it becomes increasingly inefficient as the value of u approaches $n/2$. Figure 6 shows this effect for $n = 3$ and $u = 1.5$. The valid tasksets lie inside the marked hexagon but the area of the plane within which tasksets are generated is 50% larger than this meaning $1/3$ of tasksets will be discarded in this case. The algorithm becomes extremely inefficient for large values of n with values of u close to $n/2$.

Davis and Burns [12] used a pragmatic discard limit of 1000 to avoid UUnifast-Discard making intractable attempts to find valid tasksets. This limit restricts the maximum number of attempts at taskset generation to 1000 times the number of tasksets required.

C. Randfixedsum Algorithm

Stafford's Randfixedsum [13] was designed to efficiently generate a set of vectors which are evenly distributed in $n-1$ dimensional space and whose components sum to a constant value. The key to its efficiency is that it does not require any random samples to be rejected. It can be applied directly to the problem of task utilisation generation with a chosen constant total taskset utilisation. This algorithm was made public with an open source Matlab implementation accompanied by a document explaining the theory behind the algorithm. However, it has not been formally published before.

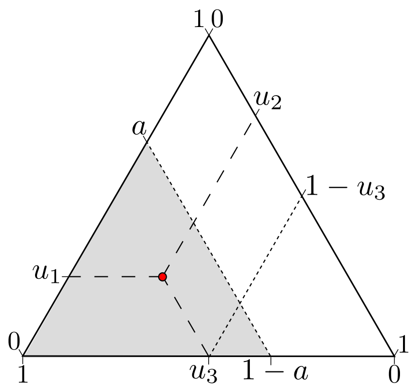


Fig. 7. Triangular axes for plotting triplets with sum 1

To explain how the Randfixedsum algorithm works, we will first turn to the case of $n = 3$ and $u = 1$. All valid tasksets with cardinality 3 and total utilisation 1 can be plotted on triangular axes, each of length 1. Such a set of axes is shown in figure 7. As noted by Stafford [13], if the points in the triangle are evenly distributed then the number of points inside any area within the triangle will be proportional to that area. A smaller triangle can be created as shown in figure 7 by drawing a line between $(a, 0, 1-a)$ and $(0, a, 1-a)$. The area of the large triangle is $\frac{\sqrt{3}}{4}$ and the area of the smaller shaded triangle is $\frac{\sqrt{3}}{4}a^2$. For the correct proportion of points to lie inside the shaded triangle compared to the whole, the probability of a point being inside the shaded triangle should be a^2. This is equivalent to requiring $P(u_3 > (1-a)) = P((1-u_3) < a) = a^2$. This can be done by selecting a uniform random value r_2 and then setting $u_3 = 1 - r_2^{1/2}$ as is done in UUniFast. Following this, a value u_2 is selected between 0 and $1-u_3$ along a line. Any segment of this line should contain a number of points proportional to its length. This is done in UUniFast by setting $u_2 = (1-u_3) - r_1$ where r_1 is another uniform random value.

Extending the concept above to several dimensions, it can be seen that UUniFast will evenly distribute points inside an $n-1$ dimensional simplex. Stafford's algorithm divides up the valid region of points into multiple $n-1$ dimensional simplexes and then applies an algorithm similar to UUniFast to select points within a randomly chosen simplex. By making the probability of selecting each simplex proportional to its volume, points are evenly distributed throughout the entire valid region. The remainder of this section describes how the simplexes are generated.

To divide the valid region into simplexes, the centre point at $(u/n, u/n, \ldots, u/n)$ is chosen. From here, we select a point by moving to 1 or 0 in one of the dimensions and then move to the centre of the boundary that was hit. For example the point $(0, u/(n-1), \ldots, u/(n-1))$ or the point $(1, (u-1)/(n-1), \ldots, (u-1)/(n-1))$. This is done repeatedly until we reach a point where the sum of 0s and 1s is exactly $k = \lfloor u \rfloor$. At this stage, if another 0 or 1 is selected, then the only way to maintain the constant sum is to pick a point outside the valid

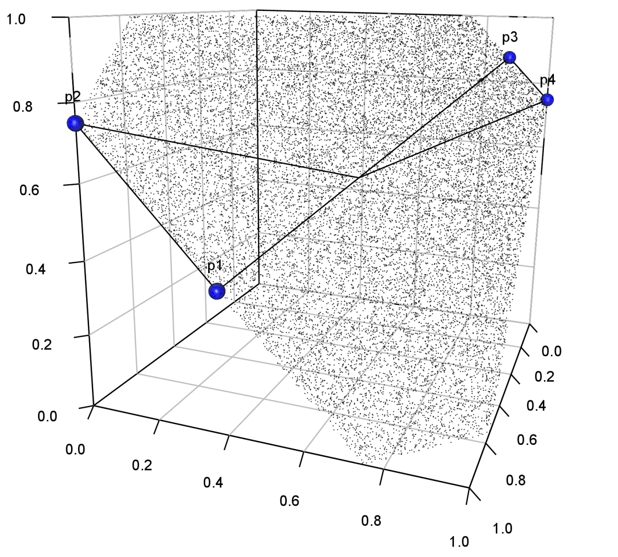


Fig. 8. Two types of simplex generated by Stafford's algorithm for $n = 3$, $u = 1.75$

region. The 1s can be selected for any k of $n - 1$ dimensions and the sequence of points (including the initial centre point) used to construct the simplex can be ordered in $n!$ ways. This creates $\binom{n-1}{k}n!$ different simplexes.

Figure 8 shows the $\binom{2}{1} = 2$ types of simplex for $n = 3$ and $u = 1.75$. $3! = 6$ of each type of simplex are needed to cover the entire valid region. Stafford's algorithm calculates the hypervolume of each type of simplex and uses this for its probability of selection. Points are then evenly distributed inside each simplex. The final stage of Stafford's algorithm is to randomly permute the order of dimensions within each point to get coverage of the whole valid region. Stafford's algorithm is available online [13] written in the Matlab language. We aim to implement the algorithm within a taskset generation tool which will be made publicly available.

V. Conclusion

The research described in this paper was motivated by the need for taskset generation algorithms to support the study of scheduling algorithm and schedulability test effectiveness for multiprocessor real-time systems.

The main contributions of the paper are as follows:

- Investigation of how sampling periods from uniform and log-uniform distributions affects the schedulability of tasksets running on a single processor using fixed priority scheduling.
- The application of Stafford's Randfixedsum algorithm to the selection of task utilisation values for tasksets with a total utilisation greater than 1. This algorithm generates an unbiased distribution of task utilisation values, and is capable of doing so for any valid values of taskset utilisation and taskset cardinality.

If the experimental region of interest is where the total taskset utilisation is either very small or large compared to the taskset cardinality then UUniFast-Discard is efficient is simple to implement. As the taskset utilisation approaches $n/2$ from either above or below, the algorithm is much less efficient and impractical for larger tasksets.

The existing Matlab implementation of Randfixedsum is highly efficient in all regions of the parameter space. We therefore recommend its use in multiprocessor taskset generation. We aim to make implementations of this algorithm in other languages available shortly.

Acknowledgments

The authors would like to thank Enrico Bini for his contribution to discussions about the applicability of the UUniFast algorithm to the multiprocessor case. The research was funded in part by the EPSRC Tempo project.

References

[1] J. Carpenter, S. Funk, P. Holman, A. Srinivasan, J. Anderson, and S. Baruah, *A Categorization of Real-time Multiprocessor Scheduling Problems and Algorithms*. Chapman & Hall/CRC, 2004, ch. 30.

[2] B. Andersson, S. Baruah, and J. Jonsson, "Static-priority scheduling on multiprocessors," in *Proceedings of the 22nd IEEE Real-Time Systems Symposium*, 2001, pp. 193–202.

[3] B. Kalyanasundaram and K. Pruhs, "Speed is as powerful as clairvoyance," in *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, 1995, pp. 214–221.

[4] E. Bini and G. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Systems*, vol. 30, no. 1-2, pp. 129–154, May 2005.

[5] J. Lehoczky, L. Sha, and Y. Ding, "The rate monotonic scheduling algorithm: exact characterization and average case behavior," in *Proceedings of the 10th Real Time Systems Symposium (RTSS 1989)*, 1989, pp. 166–171.

[6] R. I. Davis, A. Zabos, and A. Burns, "Efficient exact schedulability tests for fixed priority real-time systems," *IEEE Trans. Comput.*, vol. 57, no. 9, pp. 1261–1276, 2008.

[7] F. Zhang and A. Burns, "Schedulability analysis for real-time systems with edf scheduling," *IEEE Transactions on Computers*, vol. 58, no. 9, pp. 1250–1258, 2009.

[8] M. Bertogna, "Real-time scheduling for multiprocessor platforms," Ph.D. dissertation, Scuola Superiore SantAnna, Pisa, 2007.

[9] M. Bertogna and M. Cirinei, "Response-time analysis for globally scheduled symmetric multiprocessor platforms," in *Proceedings of the 28th IEEE International Real-Time Systems Symposium*. IEEE Computer Society, 2007, pp. 149–160.

[10] M. Bertogna, M. Cirinei, and G. Lipari, "Schedulability analysis of global scheduling algorithms on multiprocessor platforms," *IEEE Trans. Parallel Distrib. Syst.*, vol. 20, no. 4, pp. 553–566, 2009.

[11] T. Baker, M. Cirinei, and M. Bertogna, "Edzl scheduling analysis," *Real-Time Systems*, vol. 40, no. 3, pp. 264–289, December 2008.

[12] R. I. Davis and A. Burns, "Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems," in *Proceedings of the 30th IEEE Real-Time Systems Symposium (RTSS 2009)*, December 2009, pp. 398–409.

[13] R. Stafford. (2006) Random vectors with fixed sum. [Online]. Available: http://www.mathworks.com/matlabcentral/fileexchange/9700

[14] A. Burns and G. Baxter, "Time bands in systems structure," in *Structure for Dependability: Computer-Based Systems from an Interdisciplinary Perspective*. Springer London, 2006, pp. 74–88.

[15] J. W. Liu, *Real-Time Systems*. Prentice Hall, 2000.

[16] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. J. Wellings, "Applying new scheduling theory to static priority pre-emptive scheduling," *Software Engineering Journal*, vol. 8, pp. 284–292, 1993.

[17] P. Hall, "The distribution of means for samples of size n drawn from a population in which the variate takes values between 0 and 1, all such values being equally probable," *Biometrika*, vol. 19, no. 3/4, pp. 240–245, 1927. [Online]. Available: http://dx.doi.org/10.2307/2331961

[18] H. E. Daniels, "Saddlepoint approximations in statistics," *The Annals of Mathematical Statistics*, vol. 25, no. 4, pp. 631–650, December 1954.

[19] J. L. Jensen, *Saddlepoint Approximations*. Oxford University Press, May 1995.

A Statistical Approach to Simulation Model Validation in Timing Analysis of Complex Real-Time Embedded Systems

Yue Lu, Johan Kraft, Thomas Nolte and Christer Norström
Mälardalen Real-Time Research Centre
Mälardalen University, Västerås, Sweden
{yue.lu, johan.kraft, thomas.nolte, christer.norstrom}@mdh.se

Abstract

Simulation-based analysis methods make few restrictions on the system design and scale to very large and complex systems, therefore they are widely used in timing analysis of complex industrial embedded systems. This paper presents a statistical approach to validation of temporal simulation models extracted from complex real-time embedded systems, by introducing existing mature statistical methods to the context. The proposed approach first collects sampling distributions of response time and execution time data of tasks in both the modeled system and the model, based on simple random samples (SRS). The second step of the approach is to compare the sampling distributions, regarding interesting timing properties, by using the non-parametric two-sample Kolmogorov-Smirnov test. The evaluation using a fictive system model inspired by a real robotic control system with a set of change scenarios, shows a promising result. The proposed algorithm can identify temporal differences between the target system and its extracted model, i.e., the algorithm can assess whether the extracted model is a sufficiently accurate approximation of the target system.

1 Introduction

To date, most existing embedded real-time software systems have been developed in a traditional code-oriented manner, over extended periods of time, sometimes spanning decades. As a result, many such systems become large and increasingly complex. Further, to maintain, verify and reuse these systems is difficult and expensive. There are many industrial embedded systems having a very complex runtime behavior, due to that they are highly configurable and event-triggered. Such systems consist of millions of lines of C code, and contain 50 - 100 tasks or more, out of which many tasks have real-time constraints. One example of such systems is the robotic control systems developed by ABB [1]. Further, the temporal dependencies between tasks in such systems vary the execution time and response time of tasks radically. We refer to such systems as *Complex Real-Time Embedded Systems* (CRTES).

Simulation-based analysis of CRTES has the potential of not only allowing for response-time analysis of such systems [2], [3], but also facilitating migration toward a component based real-time system by e.g., analyzing the timing properties of the existing code and wrapping it into components. Moreover, simulation-based methods can also be used in timing impact analysis [4], i.e. to analyze the impact of changes on a system's temporal behavior, before introducing changes to the system.

A major issue when using simulation-based timing analysis is how to obtain the necessary analysis model, which should be a subset of the original software program focusing on behavior of significance for task scheduling, communication and allocation of logical resources. For many systems, manual modeling would be far too time-consuming and error-prone. Two methods for automated model extraction are proposed in [5]. A tool for automated model extraction 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 program slicing [6]. The output of MXTC is simulation models for the RTSSim simulation framework [7].

However, there is one important issue to be raised, i.e. *model validity*, which is defined as *the process of determining whether a simulation model is an accurate representation of the system, for the particular objectives of the study* [8]. As a model is an abstraction of the system, some system details may be omitted in the model, for instance when using probabilistic execution time modeling. Thus, the results from a simulation of such models may not be identical to the recordings of the system, e.g., with regard to the exact task response time. In order to convince system experts to use simulation-based methods, the models should reflect the system with a satisfactory level of significance, i.e., as a sufficiently accurate approximation of the actual system. Moreover, other threats to model validity are the configuration of the model extraction tool and bugs in the model extraction and analysis tools. Therefore, an appropriate validation process should be performed before using the models.

In this paper, we present a statistical approach for validation of temporal simulation models extracted from real industrial control systems containing intricate task execution dependencies. That is, to consider this particular problem as a statistical problem, then, which could be solved by using existing, mature methods from the field of statistics.

The proposed method *StatiVal* collects sampling distributions by combining using simple random samples (SRS) [9] with our presented mechanism to eliminate dependencies among raw Response Time (RT) and Execution Time (ET) data caused by task execution dependencies in the system. Next, our method will produce results concerning whether the model is a sufficiently accurate approximation of the target system, from the perspective of relevant timing properties such as response time and execution time of tasks in the modeled system and the extracted model, by using the non-parametric two-sample Kolmogorov-Smirnov test [10]. Since our tool for model extraction (MXTC) is not yet ready, in this work, we evaluate StatiVal by using a manually created simulation model inspired by an industrial robotic control system. Then, the original model is compared with different variants of the model, each of which variant corresponds to a particular change scenario. Our evaluation of this method shows the promising results, i.e., StatiVal can identify timing differences between the modeled system and models, and should be applicable in a non-trivial industrial evaluation and deployment of our framework for simulation-based analysis.

The remaining part of the paper is organized as follows: Section 2 introduces the simulation model used in this work. Section 3 presents the related work about model validation at first, and then gives problem formulation, descriptive statistics of raw RT and ET data of tasks in the evaluation model, and the problems with using parametric statistics, respectively. Section 4 and Section 5 introduce our proposed method and evaluation results, and finally, Section 6 concludes the paper and discusses future work.

2 RTSSim Simulation Models

The proposed validation method primarily targets simulation models for the RTSSim simulation framework, which is quite similar to *ARTISST* [11] and *VirtualTime* [12]. 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, interprocess communication (message queues) and synchronization (semaphores). The default scheduling policy of RTSSim is Fixed-Priority Preemptive Scheduling (FPPS) and each task has scheduling attributes such as priority, period, offset and jitter. RTSSim allows for three types of selections which are directly controlled by simulator input data: Selection of execution times in `execute` statements; Selection of task jitter; Selection of task behaviors, depending on the system environment, e.g., random number of external events generated by sensors. In RTSSim, Monte Carlo simulation is realized by providing randomly generated input data. A more thorough description of RTSSim can be found in [7].

3 Model Validation

3.1 Related Work

For the sake of space, we only briefly introduce the related work concerning the model validation process. There are various methods to do the comparison; these methods are either objective or subjective. Subjective methods are often used for validation of simulation models; examples of subjective methods are Face Validation, Graphical Comparisons and Sensitive Analysis [13], which are highly dependent on domain expertise and hence error-prone. Objective methods use mathematical methods to compare outputs from the real system with output from the simulation model. In [14], the authors presented a notation of model equivalence based on observable property equivalence which is used to compare results of a model and an actual system. A method in [15] is presented for automated validation of models extracted from real-time systems by checking if the model can generate the same event sequences as the recorded event sequences from the system using a model checker.

3.2 Problem Formulation

We are given a model $S^{'}$ which is extracted from a real system (or modeled system) S containing a task set Γ including n tasks, where $n \in \mathbb{N}$. Let $RT_{samples}(S^{'}, \tau_i)$, $RT_{samples}(S, \tau_i)$, $ET_{samples}(S^{'}, \tau_i)$ and $ET_{samples}(S, \tau_i)$ denote the sampling distributions of the response time and execution time measured for a task τ_i in $S^{'}$ and S respectively. The goal of the problem is then to find: whether there are statistically significant differences between the system and model distributions with respect to response times and execution times of the adhering tasks, or can they be considered statistically equal (i.e., from the same population).

3.3 Descriptive Statistics of Raw RT and ET Data

Table 1 shows the numerical summary of the center and the spread (or variability) of sampling distributions of the response time (RT) data of tasks in Model 1 (M1) containing intricate execution dependencies, used for the evaluation in Section 5. In Table 1, *Std. Dev*, *Q1* and *Q3* represents *standard deviation*, *first quartile* and *third quartile* of the sampling distribution respectively. As we can see, the skewness of sampling distributions for all the tasks except for the IO task are right (positive) skewed (i.e., the numerical representation of tasks' skewness are positive; in the view of graph, the sampling distribution has relatively few high values, and the mass of the distributions is concentrated on the left of the figure). Further, the outliers existing in raw RT data as well as ET data of all tasks cannot be removed since they are not generated due to system errors or hardware failures. Therefore, we have the reasoning to add the *five-number summary* introduced in [9] consisting of *Min*, *Q1*, *Median*, *Q3* and *Max* to Table 1. Due to limited space, we only show the sampling distribution of raw RT data of one task i.e., the CTRL task when the number of samples is large enough i.e. 199 990 in one simulation run (refer to row `Samples` for the CTRL task in Table 1), as an example shown in Figure 1. Further, note that the outliers in the picture might not be clear enough to see, though in fact, they approximately exist in the range of [3 000, 6 829] along with the horizontal axis.

Table 1. Descriptive statistics of sampling distributions of raw RT data of tasks in the system model M1 used in the evaluation.

	DRIVE	*IO*	*CTRL*	*PLAN*
Samples	199994	400000	199990	199988
Mean	222.08	125.0	1967.3	2002.9
Std. Dev	14.291	45.576	389.98	412.46
Skewness	6.7334	0.00128	0.38184	7.0644
Min	220	0	1024	332
Q1	220	100	1594	1631
Median	220	125	1919	1931
Q3	220	150	2339	2376
Max	420	250	6829	45957

3.4 Dependencies between Raw RT and ET Data of Tasks

In our case, due to intricate task execution dependencies in the system, an upcoming RT data may not be independent with the RT data previously recorded at each simulation run (we refer to such RT and ET data as *raw RT and ET data*). The same problem applies for raw ET data. Second, in the conventional statistical procedure (*parametric test*), e.g., t-test, analysis of variance (ANOVA) [16], one important assumption is that the underline population is assumed to follow a normal distribution. However, such assumption cannot be made since the sampling distribution of either raw RT data or raw ET data of all tasks often is conforming to a multimodal distribution having several peaks (consider Figure 1 as an example). Specifically, because of such distinctive feature of our target industrial control system, it is difficult to bring conventional statistical methods into the context. A new way of constructing the sampling distributions of tasks' RT and ET data has to be introduced, in order to fulfill the basic requirement given by *probability distribution*, i.e. the variable described by a probability distribution is a `random variable`, of which value is a function of the outcome of a statistical experiment that has outcomes of equal probability. We will present the proposed mechanism in the following Section 4.2.

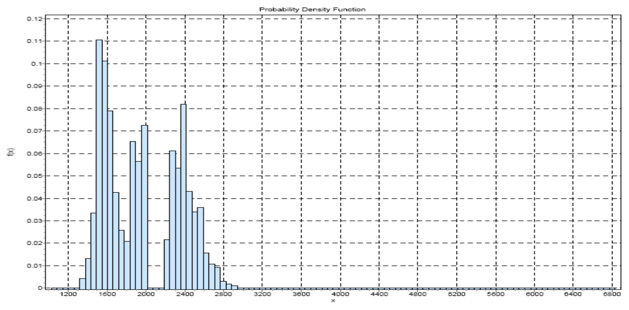


Figure 1. The sampling distribution of raw RT data of the CTRL task in the evaluation model M1.

4 Algorithm

4.1 Simple Random Samples

In order to eliminate bias on the sampling, which is a key issue of selecting samples from the population of all individuals concerning the desired information, the technique of simple random samples (SRS) [9] is adopted. SRS gives every possible sample of a given size the same chance to be chosen. For instance, Monte Carlo simulation is used as a way of implementing SRS to collect sampling distributions of RT and ET data of tasks in the extracted RTSSim model. This is done by an embedded random number generator `rnd_inst()` in the RTSSim simulator, which is an improved version of the Pseudo-random number generator used in C, i.e., `rand()` in Algorithm 1. The detailed implementation of `rnd_inst()` is shown in Algorithm 1. Moreover, empirical results showed that the distribution of ran-

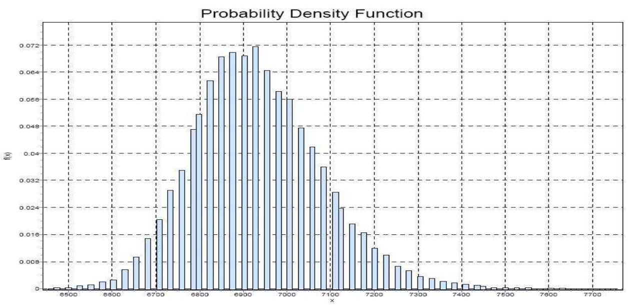


Figure 2. A new reconstructed sampling distribution of RT data of the CTRL task in the evaluation model M1.

dom numbers given by `rnd_inst()` is conforming to the uniform distribution, which assures that for each selection in RTSSim input data, all possible values in any range are equally likely to be chosen. Analogously, the sampling distributions of RT and ET data of tasks in the real system can be collected based on measurements given a randomized system input. Some of the outliers (extreme values) which are caused, e.g. hardware failure or system errors, have to be removed from the sampling distributions.

Algorithm 1 *rnd_inst*()

```
1: $temp1 \leftarrow rand()$
2: $temp2 \leftarrow rand()$
3: $ret \leftarrow temp1 \times 32768 + temp2$
4: return $ret$
```

4.2 Reconstruction of New RT and ET Sampling Distribution

In order to eliminate dependencies between raw RT and ET data of tasks due to intricate task temporal dependencies, we propose a method by first running N Monte Carlo simulations conforming to SRS as introduced previously. Further, for each task in the task set Γ, the highest value of m samples RT data and m samples ET data recorded by each simulation, will be chosen to construct new sampling distributions of RT data and ET data. By doing this, the new constructed sampling distributions of RT and ET data of tasks can be considered from a random variable, since there are no dependencies between any maximum value of RT and ET data of tasks between two independent simulations. In other words, task intricate temporal dependencies are kept in new sampling distributions of RT and ET data, while the dependencies between any RT data and ET data are eliminated. Refer to Figure 2 as an example.

4.3 Problems with Using Parametric Statistics

So as to determine if the conventional statistical procedure (*parametric test*), e.g., t-test, ANOVA, can be applied to infer parameters of new tasks' RT and ET sampling distributions used for validation purpose, the conclusion, that if such sampling distributions[1] are from a normal distribution, has to be drawn at first. In this work, it is done by using a commercial statistic analysis software *EasyFit* [17], according to the results given by a Goodness of Fit (GOF) test, i.e., Chi-squared test at α-value of 0.05[2]. The obtained results clarify that new sampling distributions of RT and ET data of all tasks do not conform to any of the 65 known distributions, e.g., Normal, Uniform, Student's t, Lognormal. The null and alternative hypotheses used in Chi-squared test, at significance level 0.05, are as follows.

1. H_0: the sampling distribution concerning the RT or ET data of task τ_i follows a specific distribution;
2. H_a: the sampling distribution concerning the RT or ET data of task τ_i does not follow a specific distribution.

Note that the 65 known distributions can be found in [17]. Further, in t-test, the mean value μ_0 of the population has to be known beforehand, which is not the fact in our case. Because a parametric test cannot be reasonably applied in this work, we thereby use the two sample Kolmogorov-Smirnov (hereafter KS test) which is non-parametric and makes no assumptions on the underline population of a sampling distribution.

4.4 StatiVal

The proposed method, *StatiVal*, is shown in Algorithm 2. The algorithm returns the result concerning if there exist a statistically significant difference between the two data sets that are from the modeled system S and the model $S^{'}$, in the view of system timing properties including tasks' response time and execution time. Further, in this work, since we cannot perform the validation between the real modeled system and the extracted model, we will instead compare a system model S inspired by a real industrial robotic control system (considered as the modeled system) with a set of models $S^{'}$ where a specific change scenario (as shown in Table 3) is applied. Both of S and $S^{'}$, are in this case simulation models, analyzed using Monte Carlo simulation which in Algorithm 2 is modeled as a function, *MTC*, with four parameters: m - the number of samples drawn from each simulation trace, τ_k - the task on focus in KS test, *Property* - either RT or ET of the task τ_k and *rnd_inst*() - a random number generator in RTSSim simulator. When the reference for comparison is a real system, the sampling distribution is built by using random measurement (e.g., by randomizing inputs to the system) at first, and then removing

[1]In our case, the number of samples i.e., 20 000 in sampling distributions of RT and ET data of tasks is statistically enough to represent the underline population.

[2]$\alpha = 0.05$ means that we are requiring that the RT and ET data of tasks give evidence against H_0 so strong that it would happen no more than 5% of the time when H_0 is true.

outliers from the sampling data that are caused by hardware failure or system errors during each system runtime observation, and finally, choosing the highest value of RT and ET data of tasks in the system. Further, because such activity is also application-specific, we therefore will not discuss it in details in this work. The outline of StatiVal is as follows:

1. Construct the sampling distribution of N RT and ET data of all the tasks in both the system S and the model S' by Monte Carlo simulation `MTC()` respectively (refer to lines 1 to 16 in Algorithm 2).
2. Use KS test to compare if sampling distributions of RT and ET data of each task τ_k in the task set Γ in both S and S' are statistically significant iteratively. If the result given by KS test is H_a, then Algorithm 2 draws the conclusion C_1, i.e. *the model S' is not a sufficiently accurate approximation of the system S due to an improper model extraction process*, and finally, stops the validation process; Otherwise, the entire validation process will terminate after all the tasks are evaluated by KS test (refer to lines 18 to 33 in Algorithm 2). In practice, KS test is conducted by using a commercial software *XLSTAT* [18], which is a plug-in to EXCEL and returns the result by comparing two sampling distributions containing 20 000 samples per each, in a few seconds.

5 Evaluation

5.1 The Evaluation Model

Currently, we are not able to perform the model validation process concerning the extracted model and a real system. Therefore, in this work, we examine the idea by using a simulation model Model 1 (M1) describing a fictive, representative industrial robotic control system developed by ABB. It is designed to include some behavioral mechanisms from the ABB system:

1. tasks with intricate dependencies in temporal behavior due to Inter-Process Communication (IPC) and globally shared state variables;
2. the use of buffered message queues for IPC, which vary the execution time of tasks dramatically;
3. although FPPS is used as base, one task, i.e., the CTRL task, changes its priority during runtime, in response to system events.

Further, the task model is presented in Table 2. The details of the model are described in [7].

5.2 Change Scenarios and Results

The RT and ET data of tasks produced by the original simulation model M1 is used as reference, for comparing the impact of a set of *change scenarios* which are initially introduced in [19] and outlined in Column *Changes Description* in Table 3. Moreover, for Case 4, 5 and 6, there

Algorithm 2 $StatiVal(\Gamma)$

```
 1: for all $\tau_k$ such that $1 \le k \le n$ in $\Gamma$ in both $S$ and $S'$ do
 2:     for all $i$ such that $1 \le i \le N$ do
 3:         $X_i \leftarrow x_{i,1}, ..., x_{i,j}, ..., x_{i,m} \leftarrow MTC(m, \tau_k, RT, rnd\_inst())$
 4:         $X_{\tau_k,i} \leftarrow Max(X_i)$
 5:         $Y_i \leftarrow y_{i,1}, ..., y_{i,j}, ..., y_{i,m} \leftarrow MTC(m, \tau_k, ET, rnd\_inst())$
 6:         $Y_{\tau_k,i} \leftarrow Max(Y_i)$
 7:         $X'_i \leftarrow x'_{i,1}, ..., x'_{i,j}, ..., x'_{i,m} \leftarrow MTC(m, \tau_k, RT, rnd\_inst())$
 8:         $X'_{\tau_k,i} \leftarrow Max(X'_i)$
 9:         $Y'_i \leftarrow y'_{i,1}, ..., y'_{i,j}, ..., y'_{i,m} \leftarrow MTC(m, \tau_k, ET, rnd\_inst())$
10:         $Y'_{\tau_k,i} \leftarrow Max(Y'_i)$
11:     end for
12:     $X_{\tau_k} \leftarrow X_{\tau_k,1}, ..., X_{\tau_k,i}, ..., X_{\tau_k,N}$
13:     $Y_{\tau_k} \leftarrow Y_{\tau_k,1}, ..., Y_{\tau_k,i}, ..., Y_{\tau_k,N}$
14:     $X'_{\tau_k} \leftarrow X'_{\tau_k,1}, ..., X'_{\tau_k,i}, ..., X'_{\tau_k,N}$
15:     $Y'_{\tau_k} \leftarrow Y'_{\tau_k,1}, ..., Y'_{\tau_k,i}, ..., Y'_{\tau_k,N}$
16: end for
17: $ret \leftarrow 0$
18: for all $\tau_k$ such that $1 \le k \le n$ in $\Gamma$ in both $S$ and $S'$ do
19:     $ret \leftarrow kstest(X_{\tau_k}, X_{\tau'_k}, \alpha)$
20:     if $ret = H_0$ then
21:         $ret \leftarrow C_0$
22:     else
23:         $ret \leftarrow C_1$
24:         return $ret$
25:     end if
26:     $ret \leftarrow kstest(Y_{\tau_k}, Y_{\tau'_k}, \alpha)$
27:     if $ret = H_0$ then
28:         $ret \leftarrow C_0$
29:     else
30:         $ret \leftarrow C_1$
31:         return $ret$
32:     end if
33: end for
34: return $ret$
```

is a DUMMY task added to the model S' with different priorities, execution times and periods (denoted as C and T in Table 3 respectively). Finally, we compare the outputs against the original model to investigate the performance of the method. The results given by StatiVal are shown in Table 3, which are in line with the expected results in [19]. More importantly, our evaluation shows a promising result, i.e. the proposed algorithm can identify temporal differences between the target system and its extracted model by showing the evidence whether the extracted model is a sufficiently accurate approximation of the target system.

6 Conclusions and Future Work

This paper has presented our work on validation of temporal simulation models extracted from real industrial control systems containing intricate task execution dependencies. In particular, we have presented and evaluated the method by using a fictive system model inspired by a real

Table 2. Tasks and task parameters for M1. The lower numbered priority is more significant, i.e., 0 stands for the highest priority.

Task	*Period (μs)*	*Offset (μs)*	*Priority*
DRIVE	2000	12000	2
IO	5000	500	5
CTRL	10000 or 20000	0	6 or 4
PLAN	40000	0	8

system with a set of change scenarios, which shows that the proposed method has the potential to identify temporal differences between the modeled system and the extracted models. As part of future work, an effort will be spent on evaluating more scenario changes on the evaluation model. Moreover, we will evaluate the method on real systems.

References

[1] "Website of ABB Group," www.abb.com.

[2] J. Kraft, Y. Lu, C. Norström, and A. Wall, "A metaheuristic approach for best effort timing analysis targeting complex legacy real-time systems," in *RTAS 08*, April 2008, pp. 258–269.

[3] M. Bohlin, Y. Lu, J. Kraft, P. Kreuger, and T. Nolte, "Simulation-based timing analysis of complex real-time systems," in *RTCSA 09*, August 2009, pp. 321–328.

[4] J. Andersson, J. Huselius, C. Norström, and A. Wall, "Extracting simulation models from complex embedded real-time systems," in *ICSEA'06*. IEEE, 2006.

[5] J. Kraft, J. Huselius, A. Wall, and C. Norström, "Extracting simulation models from complex embedded real-time systems," in *Real-Time in Sweden 2007*, August 2007.

[6] M. Weiser, "Program Slicing," in *ICSE '81*. IEEE Press, 1981, pp. 439–449.

[7] J. Kraft, "RTSSim - A Simulation Framework for Complex Embedded Systems," Mälardalen University, Technical Report, March 2009.

[8] A. M. Law, "How to build valid and credible simulation models," in *WSC '08*. Winter Simulation Conference, 2008, pp. 39–47.

[9] D. S. Moore, G. P. Mccabe, and B. A. Craig, *Introduction to the practice of statistics*, 6th ed. New York, NY 10010: W. H. Freeman and Company, 2009.

[10] A. M. Law and D. M. Kelton, *Simulation Modeling and Analysis*. McGraw-Hill Higher Education, 1999.

[11] D. Decotigny and I. Puaut, "ARTISST: an extensible and modular simulation tool for real-time systems," in *ISORC '02*, 2002, pp. 365–372.

[12] "Rapita systems, www.rapitasystems.com, 2008."

Table 3. Results obtained by using StatiVal concerning different models according to change scenarios.

Change Scenarios	*Changes Description*	*RT*	*ET*	*StatiVal*
Case 1	IO: C 23 → 46	H_a	H_a	C_1
Case 2-1	PLAN: Prio 8 → 9	H_0	H_0	C_0
Case 2-2	PLAN: Prio 8 → 3	H_a	H_a	C_1
Case 3-1	PLAN: T 40 000 → 80 000	H_0	H_0	C_0
Case 3-2	DRIVE: T 2 000 → 10 000	H_a	H_a	C_1
Case 4-1	DUMMY: Prio = 7, T = 5 000, C = 25	H_a	H_0	C_1
Case 4-2	DUMMY: Prio = 7, T = 5 000, C = 50	H_a	H_0	C_1
Case 5-1	DUMMY: Prio = 1, T = 5 000, C = 25	H_a	H_a	C_1
Case 5-2	DUMMY: Prio = 1, T = 5 000, C = 50	H_a	H_a	C_1
Case 6-1-1	DUMMY: Prio = 1, T = 10 000, C = 50	H_a	H_a	C_1
Case 6-1-2	DUMMY: Prio = 1, T = 10 000, C = 100	H_a	H_a	C_1
Case 6-2-1	DUMMY: Prio = 7, T = 10 000, C = 50	H_a	H_0	C_1
Case 6-2-2	DUMMY: Prio = 7, T = 10 000, C = 100	H_a	H_0	C_1

[13] O. Balci, "How to assess the acceptability and credibility of simulation results," in *WSC '89*. New York, NY, USA: ACM, 1989, pp. 62–71.

[14] J. Andersson, A. Wall, and C. Norström, "Validating temporal behavior models of complex real-time systems," in *SERPS'04*, September 2004.

[15] J. Huselius, J. Andersson, H. Hansson, and S. Punnekkat, "Automatic generation and validation of models of legacy software," in *RTCSA '06*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 342–349.

[16] "t-test and ANOVA, http://mathworld.wolfram.com, 2010."

[17] "Easyfit, www.mathwave.com/products/easyfit.html, 2010."

[18] "Xlstat, www.xlstat.com, 2010."

[19] F. Nemati, J. Kraft, and C. Norström, "Validation of temporal simulation models of complex real-time systems," in *CORCS'08*, July 2008.

Fault Resilience Analysis for Real-Time Systems

George Lima
Department of Computer Science
Federal University of Bahia
Salvador, Bahia - Brazil

Flávia M. Nascimento
Department of Technology in Eletro-Eletronics
Federal Institute of Bahia
Salvador, Bahia - Brazil

Verônica M. C. Lima
Department of Statistics
Federal University of Bahia
Salvador, Bahia - Brazil

***Abstract*—In this paper we present a simulation-based analysis to infer the fault resilience of real-time systems. Simulation is used to favor generality, comparability and make it possible to study the system taking into consideration its overall behavior instead of dealing only with worst-case scenarios. In the proposed approach, only parts of the schedule are simulated. Tasks can be analyzed individually, which is useful since they may have different criticality levels. We show how the results collected from simulation can be analyzed for different scheduling models.**

I. Introduction

Fault tolerance is a key aspect in real-time systems and several approaches have been proposed in this area. Most of them aim at providing a means of timeliness assessment (i.e. schedulability analysis) taking into account the possibility of error occurrences. As will be seen in Section II, usually, such approaches artificially assume a given worst-case scenario for error occurrences (error pattern) and then adapt standard schedulability analysis accordingly to check if a given system is schedulable when subject to a given error pattern.

In spite of being useful, since they provide some sort of timeliness assessment, such analyses are strictly linked with the scheduling policy and fault assumptions used, which may present some shortcomings. For example, results for fixed-priority systems [1], [2], [3], [4] cannot be applied for dynamic priority ones [5], [6] and vice-versa. Even two similar analysis techniques for the same scheduling policies may be incomparable due to the different error patterns assumed [2], [3]. This means that if one is deciding to implement a given system, he/she might not be able to choose the best approach because they are not comparable. Moreover, violating the assumed error pattern does not necessarily imply system failure since the analysis is carried out based on worst-case scenarios. Also, errors are in fact random events and do not follow a predefined pattern. Thus, it would be helpful to have a tool that can **measure fault resilience**. In this paper we present such a tool, which is based on simulation. As will be seen in Sections III and IV, the considered system model is reasonably general so that one can plug different scheduling and/or recovery models into the simulation procedure. This favors comparability, as will be seen in Section VII, where a comparison from the fault resilience viewpoint of EDF and RM is given.

Common criticisms regarding simulation-based analysis include the fact that it usually takes too much time and does not cover all possible execution paths. However, they do not apply to our approach, since only a sample of possible execution paths is needed. The idea is to simulate the system during specific time windows, which are defined based on *simulation scenarios*, a concept explained in Section V. For each time window, the simulation engine, described in Section VI, finds a lower bound on the number of errors necessary to cause a time failure in the system. By doing so for a random generated sample of simulation scenarios, one can infer the fault resilience for a given task, which is very useful since they may have different criticality levels.

II. Related Work

Most work on analyzing fault tolerant real-time systems focus on analyzing if the system is schedulable when subject to a given error pattern (fault model). Several assumptions on the error patterns have been considered. For example, some approaches assume that errors are periodic in the worst case [9], [2], [10], [11] while for others a maximum number of errors per task/job is predefined [5], [6], [12], [13], [3]. Indeed, the derived equations for schedulability analysis are strongly tied to the assumed scheduling and/or fault models. Thus, results depend on the assumed models, which prevents one to compare different systems/models from fault resilience viewpoint. As illustration consider an EDF-scheduled system that can tolerate at most k faults [6] during the hyperperiod and a RM-scheduled system that assumes periodic errors[9]. Note that each error pattern will introduce an extra workload to the system related to recover, whose execution priority is determined by the scheduling policy. Thus, such approaches may not be comparable in terms of fault resilience.

Some authors have also addressed the problem of assumption coverage [14], [4], [1], whose focus is to study to what extent the assumed error pattern is violated if errors are seen as random events. Nonetheless, due to the nature of their fault model (periodic error occurrences), the violation of the assumed error pattern does not necessarily implies system failure. Unlike such approaches, we examine to what extent a given system task can cope with errors. Without assuming a specific error pattern and considering several scheduling policies, we aim at measuring the system fault resilience of tasks as independently as possible of the assumed scheduling and fault models.

III. System Model and Notation

We consider uniprocessor and preemptive real-time systems composed of n periodic tasks $\Gamma = \{\tau_1, \ldots, \tau_n\}$. Task attributes are represented by four n-tuples, which give their periods, deadlines, worst-case execution times and recovery times, respectively denoted as $\mathbf{T} = (T_1, \ldots, T_n)$, $\mathbf{D} = (D_1, \ldots, D_n)$, $\mathbf{C} = (C_1, \ldots, C_n)$ and $\bar{\mathbf{C}} = (\bar{C}_1, \ldots, \bar{C}_n)$. We assume that each task τ_i is independent of each other. Also, $C_i \leq \min(T_i, C_i)$ and $D_i \leq T_i$.

We assume that the schedulability of the system can be assessed in fault-free scenarios. Fault tolerance is provided by executing an extra code upon error detection, which can be the re-execution of the faulty task or the execution of an alternative task. If errors are detected during the recovery of τ_i, other recovery actions can be released. Note that this model is in line with most fault tolerance techniques based on temporal redundancy such as recovery blocks or exception handlers [15], which have been widely applied to real-time systems [9], [2] and can be implemented at the task level. We do not deal with errors for which spatial redundancy is required, usually implemented using a distributed/parallel architecture [16].

As tasks in Γ are periodic, each task activation is called a *job*. The k-th job of task τ_i is released at time $\phi_i + (k-1)T_i$, where ϕ_i is the phase of τ_i. For the sake of notation simplicity, we assume that $\phi_i = 0$, for all tasks in Γ, although the proposed analysis can be easily adapted to consider fixed values of $\phi_i > 0$. Aperiodic jobs are considered for error recovery only. $\bar{J}$ and $p(J)$ denote, respectively, a recovery action for job J and its priority.

Although we do not assume a particular scheduling policy, we consider both fixed-priority and dynamic-priority which includes scheduling policies such as EDF, RM or DM, according to which jobs do not change their priorities during execution, although the priority of tasks may vary. To simplify notation we define functions $\min(\mathbf{X})$ and $\max(\mathbf{X})$, which return the minimum and maximum values of any tuple $\mathbf{X}$. We also define the function $\mathtt{rand}(a, b)$, which returns an integer value according to a discrete uniform distribution in the interval $[a, b]$.

IV. Simulation Environment Overview

Figure 1 illustrates the simulation environment, which is represented by two main components, the scheduler and the error generator. While the former component follows a given scheduling policy (e.g. RM or EDF) and tries to keep the system schedulable, the goal of the latter is to generate errors so that job deadlines are missed. The higher the effort made by the error generator, the higher the resilience of the system. No particular error pattern is assumed. A role of the error generator is to derive the worst-case error patterns for each simulation. These two components are named *simulation engine*. It is worth emphasizing that this approach is generic in the sense that scheduling and recovery policies, for example, can be plugged into the simulation straightforwardly.

Unlike existing simulation-based analysis, the simulation environment in Figure 1 does not need to simulate the whole system execution (e.g. system hyperperiod), which might be too time consuming in general. The idea is to simulate specific time windows and then to derive fault resilience by

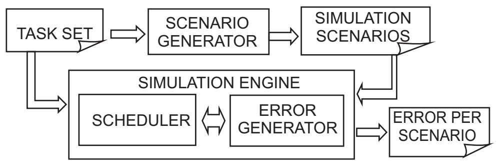


Fig. 1. Simulation Environment

statistically analyzing simulation data. These time windows are defined based on *simulation scenarios*, which are given by the scenario generator. Assuming that tasks are periodic and $C_i = \min(C_i, T_i)$, simulation scenarios are actually reference points, represented by tuples of task release times. In order to motivate the concept of simulation scenarios, consider the following example.

Example IV.1. *Consider* $\Gamma = \{\tau_1, \tau_2, \tau_3\}$ *a periodic task set, where* $\mathbf{T} = (10, 15, 20)$. *Recovery is carried out by re-executing the faulty task. Tasks must finish their execution by their deadlines even in the presence of errors.*

In this example, $h = \mathrm{lcm}(T_1, T_2, T_3) = 60$ is the task set hyperperiod. There are h/T_i simulation scenarios for task τ_i, which represents the number of its released jobs within h. We represent the set of simulation scenarios of τ_i as Ω_i. This set will be formally defined shortly. For now, we give only some intuition. Three distinct simulation scenarios for τ_1 are $(0, 0, 0)$, $(10, 0, 0)$ and $(40, 30, 40)$. More details about how to generate such values are presented in Section V. Based on each simulation scenario $\mathbf{S} \in \Omega_i$, the simulation engine determines the *simulation window*. Considering that one wishes to analyze τ_1 and $\mathbf{S} = (40, 30, 40)$ regarding Example IV.1, a simulation window could be $[20, 50)$, say. Indeed, the simulation starts at some time before $r = \min(\mathbf{S})$ so that it is possible to estimate the backlog at r. Also, it is not needed to simulate the system after time $40 + D_1$ since this is the absolute deadline of the analyzed job of τ_1 for the chosen simulation scenario.

Once a sample of simulation scenarios are randomly chosen, the corresponding simulation windows are simulated as outlined above. The main result of this simulation is the effort made by the error generator, measured as the minimum number of errors ($f_i^{\mathbf{S}}$) which makes τ_i unschedulable for a given simulation scenario $\mathbf{S}$. As will be seen, $f_i^{\mathbf{S}}$ is an approximation since it is obtained by simulation and is conservatively computed. Nonetheless, it serves well as a fault-resilience metric. After the values of $f_i^{\mathbf{S}}$ are computed, one can carry out statistical analysis to infer the fault resilience of τ_i.

Indeed, some characteristics of the proposed analysis must be highlighted such as the possibility of carrying out the analysis for different scheduling and error recovery models, which enables a comparison between different approaches. Also, the fault resilience of tasks is determined individually and based on the overall behavior of its jobs, not only worst-case. Moreover, such approach does not focus on determining whether the system is schedulable for a given error pattern. Instead, the result of the simulation represents the capacity of tasks to recover from errors.

V. Scenario Generation

The concept of simulation scenarios, as well as some useful operations have been recently discussed [17] and will be briefly presented here to make this paper self contained.

A. Simulation Scenarios

As mentioned before, simulation scenarios are defined as tuples of task release times. Since we are considering that tasks or their recoveries take their worst-case execution times to finish, this simplified representation of simulation scenarios is enough for our purpose. Nonetheless, not all tuples of release times are simulation scenarios:

Definition V.1. *Tuple* $\mathbf{S} = (S_1, \ldots, S_n)$ *is a simulation scenario of a periodic task set* $\Gamma = \{\tau_1, \ldots, \tau_n\}$ *if the following predicate holds:*

$$\mathrm{scenario}(\Gamma, \mathbf{S}) \overset{\text{def}}{=} \exists w \in \mathbf{R}, \forall S_i : (S_i + w) \bmod T_i = 0 \\ \wedge \max(\mathbf{S}) - S_i < T_i \quad (1)$$

Both conditions defined by the above predicate mean that: (a) $\mathbf{S}$ is a tuple of tasks release times; and (b) only the closest jobs, released before the last released job, are considered. For Example IV.1, it can be seen that according to Definition V.1 tuples $(0,0,0), (20,15,20)$ and $(40,30,40)$ are simulation scenarios. However, tuple $\mathbf{S} = (40,15,40)$, say, is not. Note that, although S_i is a possible release time of τ_i $(i = 1,2,3)$, the release time of τ_2 should be 30 instead of 15 to make $\mathbf{S}$ a simulation scenario for this task set example.

Consider tuples $\mathbf{S} = (20,15,20)$ and $\mathbf{S}' = (30,25,30)$, say, both scenarios for Example IV.1. Note that $S_i = S'_i + 10$ for all tuple elements S_i. This means that the same simulation effects would be observed when $\mathbf{S}$ or $\mathbf{S}'$ were simulated. In this case, it is said that $\mathbf{S}$ is equivalent to $\mathbf{S}'$. More formally:

$$\mathbf{S} \equiv \mathbf{S}' \Leftrightarrow \exists w \in \mathbf{R}, \forall i \in \{1, \ldots, n\} : S_i = S'_i + w \quad (2)$$

The time-shift operation, which returns equivalent simulation scenarios for a periodic task set Γ, is defined as:

$$\mathrm{tshift}(\mathbf{S}, w) \overset{\text{def}}{=} (S_1 + w, \ldots, S_n + w), \quad w \in \mathbf{R} \quad (3)$$

Clearly, simulation taken from equivalent simulation scenarios must be avoided. If two simulation scenarios are not equivalent, they represent possible release time distances between jobs of distinct tasks of Γ. Taking Example IV.1 for illustration and considering task τ_1, it is not difficult to see that $\Omega_1 = \{(0,0,0), (10,0,0), (20,15,20), (30,30,20), (40,30,40), (50,45,40)\}$ within the interval $[0,60)$.

Consider one of the possible simulation scenarios for Example IV.1, say $\mathbf{S} = (20,15,20)$. This scenario takes place after time advances by 20 time units from the origin. On the other hand, backtracking from $\mathbf{S}$ 10 time units, scenario $(10,0,0)$ is found. This reasoning suggests the following useful operation that gives a new scenario $\mathbf{S}'$ based on $\mathbf{S}$. More formally, $\mathbf{S}' = \mathrm{tadd}(\mathbf{S}, \Gamma, w)$, $w \in \mathbf{R}$, where $\mathbf{S}'$ is defined as

$$S'_i = S_i + \left\lfloor \frac{\max(\mathbf{S}) + w - S_i}{T_i} \right\rfloor T_i, \quad i = 1, \ldots, n \quad (4)$$

It is important to notice that the time-add operation always lead to a valid simulation scenario [17]. Also, observe that $\mathrm{tadd}(\Gamma, \mathbf{S}, w)$ is a step-function which changes its values whenever $\max(\mathbf{S}) + w - S_i$ is multiple of T_i. Further, since $h = \mathrm{lcm}(T_1, \ldots, T_n)$, it is not difficult to check that $\mathrm{tadd}(\Gamma, \mathbf{S}, w) \equiv \mathrm{tadd}(\Gamma, \mathbf{S}, w + h)$. These observations imply that there is a finite set of values for w that can be used to generate all simulation scenarios. In the following section we show how to generate random subsets of simulation scenarios.

B. Generation Procedure

Algorithm 1 is a procedure which generates a random and not biased subset Ω_i^* for a given task $\tau_i \in \Gamma$ in m steps, where $m = |\Omega_i^*|$. Each step takes a time interval of size h^*, where $m = h/h^*$ (line 2). For simplicity, we assume that all jobs are released at time $t = 0$, although any other value could be assumed. Note that for each interval the algorithm chooses a random value k which always leads to a valid scenario (line 5). Both *tadd* and *tshift* operations (lines 8 and 9), are used to bound the task release times, which reduces the complexity due to large numbers arithmetics. Indeed, the running time of Algorithm 1 depends on the value of h^*. For example, if $h^* = T_i$, Algorithm 1 works with numbers as large as T_i. On the other hand, if $h^* = h$, the algorithm has to deal with numbers as large as h. Clearly, running the algorithm several times with a big value of h^* is not recommended. One possibility is to bound the bit-size of numbers generated by the algorithm according to the target architecture to achieve a good trade-off between the number of steps and the time/memory needed for arithmetics. Since the time complexity of line 5 is $O(n \log^2 h^*)$ and this line is executed $m = \frac{h}{h^*}$ times, the time complexity of the algorithm is $O(\frac{nh}{h^*} \log^2 h^*)$.

Algorithm 1: Random generation procedure. Is is assumed that $m < h/T_i$.

```
1  $\Omega^* \leftarrow \emptyset$; $j \leftarrow 0$; $\mathbf{S} \leftarrow (0, \ldots, 0)$;
2  $h \leftarrow \mathrm{lcm}(T_1, \ldots, T_n)$; $h^* \leftarrow \lfloor h/m \rfloor$;
3  repeat
4      $k \leftarrow \mathtt{rand}(0, (h^* - T_i)/T_i)$;
5      $\mathbf{S}' \leftarrow \mathrm{tadd}(\Gamma, \mathbf{S}, kT_i)$;
6      $\Omega_i^* \leftarrow \Omega_i^* \cup \mathbf{S}'$;
7      $j \leftarrow j + 1$;
8      $\mathbf{S} \leftarrow \mathrm{tadd}(\Gamma, \mathbf{S}, h^*)$;
9      $\mathbf{S} \leftarrow \mathrm{tshift}(\mathbf{S}, -S_1)$;
10 until $(j = m)$ ;
```

Considering Example IV.1, task τ_1 and defining $m = 3$ (i.e. $h^* = 20$), the algorithm chooses a value of $k \in \{0,1\}$ in each of its steps. In the first step two possible scenarios can be generated, $(0,0,0)$ or $(10,0,0)$. Note that $\mathbf{S}$ is kept constant in each step, serving as a base scenario for generating $\mathbf{S}'$. Table I illustrates the behavior of Algorithm 1. The last column of the table indicates the equivalence between the generated scenarios $\mathbf{S}'$ and those that would be generated if line 9 was removed from the algorithm.

TABLE I
Ω_1^* FOR EXAMPLE IV.1.

step	$\mathbf{S}$	k	$\mathbf{S}'$	$\mathbf{S}' \equiv$
1st	$(0,0,0)$	0	$(0,0,0)$	$(0,0,0)$
		1	$(10,0,0)$	$(10,0,0)$
2nd	$(0,-5,0)$	0	$(0,-5,0)$	$(20,15,20)$
		1	$(10,10,0)$	$(30,30,20)$
3rd	$(0,5,0)$	0	$(0,5,0)$	$(40,45,40)$
		1	$(10,5,0)$	$(50,45,40)$

VI. SIMULATION ENGINE

In this section we describe both the simulation and the error generator procedures. Assume that a task in $\Gamma = \{\tau_1, \dots, \tau_n\}$ is to be analyzed for a specific simulation scenario $\mathbf{S} = (S_1, \dots, S_n)$. The job of this task, released at S_i, namely J_i, is called hereafter the analyzed job. Figure 2 sketches the simulation process. Scenario $\mathbf{S}$ and a previous scenario $\mathbf{S}'$ are indicated in the gray area of the figure. The first release time of jobs in $\mathbf{S}$, indicated by the vertical arrows, whose priorities are at least $p(J_i)$ is denoted as $r \leq S_i$ in the figure. This job must be considered when analyzing the effects of errors in the execution of J_i.

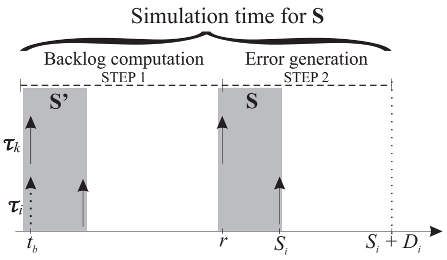


Fig. 2. Two step simulation procedure.

The simulation of the system regarding $\mathbf{S}$ involves two problems: (a) determining the execution backlog at r, which is related to jobs released before r; and (b) generating the minimum number of errors from r onwards so that the analyzed job misses its deadline. Nonetheless, exact solutions to problems (a) and (b) may be computationally too expensive. Thus, our approach to solving them is to derive an upper bound for (a) and a lower bound for (b) so that the effort of the error generator is not overestimated. The simulation procedure has two steps, as shown in Figure 2, which will now be detailed.

A. Backlog Computation

The backlog computation aims at determining the interference from jobs in a previous scenario $\mathbf{S}'$ in the analyzed job. To do so, we estimate the backlog for $\mathbf{S}$ (a) going back to a previous scenario $\mathbf{S}'$ and (b) forcing the release time of all tasks in Γ be at time $t_b = \min(\mathbf{S}')$. The remainder execution time after simulating the system within $[t_b, r)$ should give the desired upper bound. Ideally $\mathbf{S}'$ should be a scenario which gives a good trade-off between simulation time and backlog estimation. In this work, though, we follow a simple approach to computing $\mathbf{S}'$, which is going back to the closest scenario before $\mathbf{S}$. In experiments we have also used others scenarios but on average the effects on the backlog was not significant.

Once t_b is computed, the simulation starts executing the jobs released in $[t_b, r)$ but with the error generator deactivated. Since some jobs are artificially released at t_b, as illustrated by the dotted-arrowed line in Figure 2, there could be an execution overload in $[t_b, r)$ which are generated for the purpose of backlog estimation only. In order to reduce this artificial overload, the jobs that are executed in $[t_b, r)$ until their deadlines are missed, time at which they are discarded. This is done to reduce the pessimism of the simulation-based analysis. Jobs in $[t_b, r)$ are called *backlog jobs*.

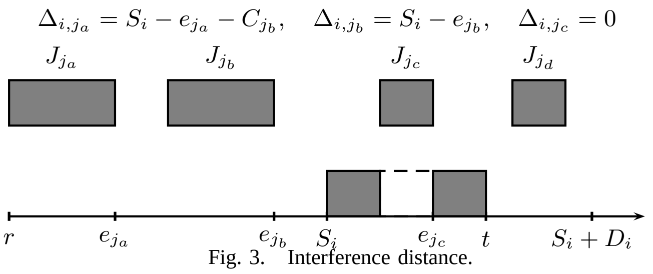
B. Error Generation

The simulation during $[r, S_i + D_i)$ is carried out with the error generator active. The strategy is to generate errors in the job which causes the highest interference in the analyzed job J_i. As the goal is to estimate a lower bound on the minimum number of generated errors that make J_i miss its deadline, faulty jobs are allowed to execute beyond their deadline (except for J_i). In other words, the optimization problem of determining which jobs fail during simulation is circumvented. According to this approach the found number of errors is guaranteed not to be overestimated but can be underestimated.

Consider a scenario $\mathbf{S}$ and a time interval $[r, t), r < t < S_i + D_i$ in which J_i is active. The set of all jobs that may interfere in J_i during $[r, t)$ is defined as $hp_i^S(r, t)$. Indeed, since the error generator must not let J_i meet its deadline, every time t at which J_i would successfully finish its execution an error must be generated in a job $J_j \in hp_i^S(r, t)$ which causes the greatest interference in J_i. In order to find out J_j, we define the concept of *interference distance* ($\Delta_{i,j}^S$), illustrated in Figure 3.

Let e_j be the finishing time of J_j when no errors take place and $C_k(t)$ the pending worst-case execution cost at time t of any $J_k \in hp_i^S(r, t)$. Observe that time t is a possible successful finishing time of J_i and so the error generator must generate an error in some job in $hp_i^S(r, t)$ so as to prevent J_i from finishing. There are three possibilities denoted in Figure 3 as J_{j_a}, J_{j_b} and J_{j_c}. Note that J_{j_d} is only active after t and so it is not considered as an option. Also, note that J_{j_c} was released after S_i and before t. In this case, $\Delta_{i,j_c}^{\mathbf{S}} = 0$ and so any error in J_{j_c} would cause an extra interference in the execution of J_i. This is not true for jobs J_{j_a} and J_{j_b}. Indeed, J_i would suffer interference of these jobs due to errors only if their recovery times are greater than their interference distances, Δ_{i,j_a} and Δ_{i,j_b}, respectively.

In the example no previous errors were considered. Now consider a general case where the error generator is to generate an additional error and $f_i^{\mathbf{S}}$ errors have already been generated. In this case, the error may be generated in (a) $J_j \in hp_i^S(r, S_i)$ or (b) $J_j \in hp_i^S(S_i, t)$. The maximum interference between jobs in (a) and (b) gives the desired lower bound on the number


$\Delta_{i,j_a} = S_i - e_{j_a} - C_{j_b}$, $\Delta_{i,j_b} = S_i - e_{j_b}$, $\Delta_{i,j_c} = 0$
J_{j_a}
J_{j_b}
J_{j_c}
J_{j_d}
r
e_{j_a}
e_{j_b}
S_i
e_{j_c}
t
$S_i + D_i$
Fig. 3. Interference distance.

of generated errors whenever $f_i^{\mathbf{S}} + 1$ errors make J_i miss its deadline.

It is important to emphasize that this strategy can be used because we are considering that a chosen faulty job J_j is assumed to execute beyond its deadline $S_j + D_j$. This is done for the sake of analysis only and does not imply that we are restricting the system task model. It is clear that since J_j has the highest interference in the execution of J_i according to this assumption, any other combination of faulty jobs cannot cause a greater interference. Therefore, we are conservatively determining the error generator effort as mentioned before.

C. Simulation Procedure

Algorithm 2: Simulation engine

```
1  $t' \leftarrow S_i + D_i$; $r \leftarrow \min_{J_k \in \mathrm{hp}_i^S}(S_k)$;
2  $\mathbf{S}' = \mathrm{tadd}(\Gamma, \mathbf{S}, -\min(\mathbf{T}))$;
3  $t_b \leftarrow \min_{J_k \in \mathrm{hp}_i^{S'}}(S'_k)$; $t \leftarrow t_b$;
4  foreach $J_k \in \mathrm{hp}_i^S(t_b, t')$ do
5      enqueue$(k, C_k, p(J_k))$;
6  enqueue$(0, t - t', p(J_i) - 1)$;                /* a dummy job */;
7  $f_i^S \leftarrow 0$; $\bar{C} \leftarrow 0$;
8  while $(t \leq t')$ do
9      $(k, C, p) \leftarrow$ dequeue$(t)$;
10     $s \leftarrow$ nextJob$(t, p)$;
11     if $t + C \leq s$ then                         /* $J_k$ finishes */
12         $t \leftarrow t + C$;
13         if $t < r$ then                             /* backlog job */
14             if $S_k + D_k > t + C$ then
15                 $t \leftarrow S_k + D_k$;
16         else                        /* error generator active */
17             if $j = i \wedge t \leq S_i + D_i$ then
18                 $f_i^S \leftarrow f_i^S + 1$;
19                 $x \leftarrow \max_{J_j \in \mathrm{hp}_i^S(r, S_i)}(f_i^S \bar{C}_j - \Delta_{i,j}^S)$;
20                 $y \leftarrow \max_{J_j \in \mathrm{hp}_i^S(S_i, t)}(\bar{C}_j)$;
21                 if $x > \bar{C} + y$ then
22                     enqueue$(i, x - \bar{C}, p(J_i))$;
23                     $\bar{C} \leftarrow x$;
24                 else
25                     enqueue$(i, y, p(J_i))$;
26                     $\bar{C} \leftarrow \bar{C} + y$;
27
28     else                                       /* $J_k$ is preempted */
29         enqueue$(k, C - (s - t), p)$;
30         $t \leftarrow s$;
31
```

Algorithm 2 implements the simulation engine. It receives as input parameters a task set Γ, one of its simulation scenarios $\mathbf{S}$, and a task to be analyzed regarding $\mathbf{S}$, whose job is released at S_i. The simulation interval $[t_b, t')$ is set in lines 1-3. Variables $\bar{C}$ and $f_i^{\mathbf{S}}$ store the sum of recovery times of faulty jobs and the number of errors, respectively. The final value of $f_i^{\mathbf{S}}$ is the generated number of errors by the error generator that make J_i miss its deadline, meaning that scenario $\mathbf{S}$ is resilient to at least $f_i^{\mathbf{S}} - 1$ errors. Initially, all jobs in the simulation interval are enqueued according to their priorities and release times (lines 4-5). A dummy job is also enqueued at priority level $p(J_i) - 1$ (line 6) which is used for advancing time during idle intervals.

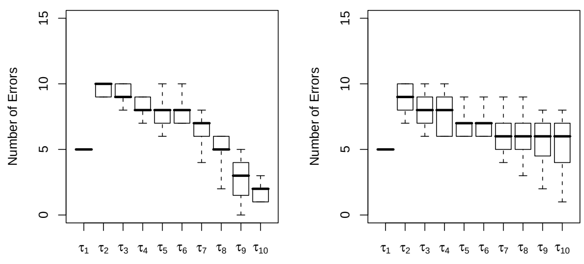
Any job is dispatched to execution at time t as follows. The highest priority ready job at t is dequeued (line 9). Then the `nexJob` function returns the next release time of the job with priority at least p whose release time is greater than t. If $t + C > s$, the dispatched job J_k is executed until time s when a preemption occurs. Otherwise, there are three situations to be checked. If J_k is a backlog job, it is executed until either time $t + C$ or time $S_k + D_k$. In the former case, time is simply advanced to $t + C$. In the latter case, J_k misses its deadline at time $S_k + D_k$ and is discarded (lines 14-15). Finally, if $J_k = J_i$ and J_i meets its deadline, an additional error is generated in the job which maximizes the interference in J_i, as explained in Section VI-B. Note that the recovery time of the faulty job is added to C_i. This avoids possible backtracking to execute the recovery of J_j, simplifying the simulation.

VII. Statistical Analysis

In this section we use classical statistical inference [18] to estimate the confidence interval for the system fault resilience. We determine a $100(1 - \alpha)\%$ confidence interval for the number of errors computed by the described approach, where $1 - \alpha$ is the confidence level. In other words, as we have used a sample of simulation scenarios to determine the fault resilience of the analyzed system, we are interested now in infering the fault resilience of the system for the whole set of simulation scenarios.

First, we illustrate the analysis with an example, for which we want to determine the mean value of $f_i^{\mathbf{S}}$, for all $\tau_i \in \Gamma$ and for all $\mathbf{S} \in \Omega_i$. We considered a periodic task set with 10 tasks and $\bar{\mathbf{C}} = \mathbf{C} = (3, \ldots, 3)$ and $\mathbf{D} = \mathbf{T} = (15, 36, 39, 40, 42, 42, 45, 45, 46, 46)$.

As the sample size $|\Omega_i^*|$ is necessary to determine a confidence interval, we need to: (a) set an acceptable *sample error* for each task, which is denoted by $|\bar{f}_i^* - \bar{f}_i|$, where $\bar{f}_i^*$ and $\bar{f}_i$ stand for the mean values related to the sample Ω_i^* and to the population Ω_i, respectively; (b) define the standard deviation σ_i based on a pilot sample; and (c) determine the confidence coefficient $1 - \alpha$. We have set $|\bar{f}_i^* - \bar{f}_i| = 5 \times 10^{-3}$ and $\alpha = 5\%$. The sample size $|\Omega_i^*|$ for each task $\tau_i \in \Gamma$ was computed according to standard statiscal methods [18]. It is worth mentioning that summing up all values of $|\Omega_i^*|$ in the above example gives only 1.30% (2682 simulation scenarios) of what would be necessary if all simulation scenarios Ω_i for each task τ_i were considered. This illustrates the scalability of the proposed analysis via statistical inference.



(a) RM (b) EDF

Fig. 4. Fault resilience distribution

Using the computed sample sizes, Algorithm 1 generated the random samples of simulation scenarios. Then Algorithm 2 was carried out. The found distributions of $f_i^{\mathbf{S}}$, for all $\mathbf{S} \in \Omega_i^*$ and all $\tau_i \in \Gamma$ are shown in Figure 4 for both RM and EDF. The boxplot diagrams indicate the quartiles of the distribution as well as their minimum and maximum values.

As can be seen in the graphics, EDF has a better overall performance in terms of fault resilience. Although this behavior was expected due to the optimality of EDF in terms of schedulability, it is important to emphasize that now the difference is being measured. It is worth mentioning that τ_1 has the same fault resilience for both schedulers. Indeed, $\bar{f}_i^{\mathbf{S}^*} = 5$ for both EDF and RM. On the other hand, for all other tasks, EDF is clearly superior to RM in terms of fault resilience. Obviously, we are not considering here problems such as possible overloads caused by admission of recovery actions, which could make EDF degrade. We stress that the goal of the proposed analysis is to point out to what extent the system support errors and is not on evaluating overload or schedulability conditions.

In order to show the scalability, we considered an experiment in which 40 task sets composed of thirty tasks each were randomly generated. Periods and execution times were randomly select in the intervals $[10;800]$ and $[3;30]$, respectively. For such task sets the calculated hyperperiod was of the order 10^{15}. The proposed analysis was applied for all task sets, similarly to what was explained above: we used the sample error equal to 5×10^{-3} and $\alpha = 5\%$, the fault resilience for each system task was estimated. Table II summarizes the mean effort, which are grouped per processor utilization range. There were 10 task sets in each group. The 95% confidence intervals (CI) for each group are indicated. As expected, the higher the processor utilization the less resilient the system. Since we are carrying out the analysis for different task sets, a higher variability is present. Hence, the sample sizes necessary to give the desired sample error were slightly higher, 7.76% of the total simulation scenarios.

VIII. Conclusion

We have described an innovative simulation-based analysis technique capable of measuring fault resilience of real-time systems. Tasks can be analyzed individually and their overall behavior is taken into consideration. Results of the analysis can be used to compare different systems or their scheduling models from the fault resilience viewpoint, an issue not addressed before. Unlike usual simulation-based analysis, only small parts of the schedule is (approximately) simulated, making the simulation process cost-effective. We have shown how fault resilience information can be obtained by statistically studying the results from simulation for two traditional scheduling policies. Extensions of the proposed analysis are currently being considered. Less restrictive task models and the incorporation of probabilistic behavior for the error generator can be investigated.

TABLE II
Fault resilience estimation

	RM		EDF	
% CPU	$\bar{f}_i$	CI	$\bar{f}_i$	CI
55-65	6.35	[6.237,6.552]	6.05	[6.010,6.136]
65-75	3.01	[2.996,3.050]	3.25	[3.192,3.294]
75-85	2.33	[2.300,2.358]	2.87	[2.801,2.907]
85-95	1.95	[1.890,1.960]	2.13	[2.023,2.223]

References

[1] A. Burns, G. Bernat, and I. Broster, “A probabilistic framework for schedulability analysis,” in *3rd Intl Conference on Embedded Software*, 2003, pp. 1 – 15.
[2] G. Lima and A. Burns, “An optimal fixed-priority assignment algorithm for supporting fault-tolerant hard real-time systems,” *IEEE Trans. on Computers*, vol. 52, no. 10, pp. 1332–1346, 2003.
[3] G. Lima and A. Burns, “Scheduling fixed-priority hard real-time tasks in the presence of faults,” in *2nd LADC - LNCS 3747*. Springer-Verlag, 2005, pp. 154–173.
[4] I. Broster and A. Burns, “Random arrivals in fixed priority analysis,” in *1st Intl PARTES Workshop*, 2004.
[5] F. Liberato, R. Melhem, and D. Mossé, “Tolerance to multiple transient faults for aperiodic tasks in hard real-time systems,” *IEEE Trans. on Computers*, vol. 49, no. 9, pp. 906–914, 2000.
[6] H. Aydin, “Exact fault-sensitive feasibility analysis of real-time tasks,” *IEEE Trans. on Computers*, vol. 56, no. 10, pp. 1372 – 1386, 2007.
[7] A. Wall, J. Andersson, and C. Norstrom, “Probabilistic simulation-based analysis of complex real-time systems,” in *6th IEEE ISORC*, 2003, pp. 257–266.
[8] J. Huselius, J. Kraft, H. Hansson, and S. Punnekkat, “Evaluating the quality of models extracted from embedded real-time software,” in *14th Annual IEEE Intl Conf. and Workshops on the Engineering of Computer-Based Systems*, 2007, pp. 577–585.
[9] A. Burns, R. Davis, and S. Punnekkat, “Feasibility analysis of fault-tolerant real-time task sets,” in *8th ECRTS*, 1996, pp. 29 – 33.
[10] S. Ghosh, R. Melhem, D. Mossé, and J. S. Sarma, “Fault-tolerant rate monotonic scheduling,” *Real-Time Systems*, vol. 15, no. 2, pp. 149–181, 1998.
[11] S. Ghosh, R. Melhem, and D. Mossé, “Enhancing real-time schedules to tolerate transient faults,” in *16th IEEE RTSS*, 1995, pp. 120–129.
[12] H. Aydin, R. Melhem, and D. Moss, “Tolerating faults while maximizing reward,” in *12th ECRTS*, 2000, pp. 219 – 226.
[13] P. Meja-Alvarez, H. Aydin, D. Moss, and R. Melhem, “Scheduling optional computation in fault-tolerant real-time systems,” in *7th IEEE RTAS*, 2000, pp. 323 – 330.
[14] A. Burns, S. Punnekkat, L. Strigini, and D. R. Wright, “Probabilistic scheduling guarantees for fault-tolerant real-time systems,” in *Intl Conf. on Dependable Comp. for Critical Applic.*, 1999, pp. 361 – 378.
[15] A. Burns and A. J. Wellings, *Real-Time Systems and Programming Languages*, 3rd ed. Addison-Wesley, 2001.
[16] H. Kopetz, *Real-Time Systems Design for Distributed Embedded Applications*. Kluwer Academic Publ., 1997.
[17] G. Lima and F. Nascimento, “Simulation Scenarios: a Means of Deriving Fault Resilience for Real-Time Systems,” in *11th Brazilian WTR*, 2009.
[18] M. F. Triola, *Elementary Statistics*. Pearson, 2008.

Distributed Interactive Real-time Multimedia Applications: A Sampling and Analysis Framework

George Kousiouris, Fabio Checconi, Alessandro Mazzetti, Zlatko Zlatev, Juri Papay, Thomas Voith, Dimosthenis Kyriazis

Abstract—. **The advancements in distributed computing have driven the emergence of service-based infrastructures that allow for on-demand provision of IT assets. However, the complexity of characterizing an application's behavior, and as a result the potential offered level of Quality of Service (QoS), introduces a number of challenges in the data collection and analysis process on the Service Providers' side, especially for real time applications. The aforementioned complexity is increased due to additional factors that influence the application's behavior, such as real time scheduling decisions, percentage of a node assigned to the application or application-generated workload. In this paper, we present a framework developed under the IRMOS EU-funded project that enables the sampling and gathering of the necessary dataset in order to analyze an application's behavior. Processing of the resulting dataset is also conducted in order to extract useful conclusions regarding CPU allocation and scheduling decisions effect on the QoS. We demonstrate the operation of the proposed framework and evaluate its performance and effectiveness using an interactive real-time multimedia application, namely a web-based eLearning scenario.**

I. Introduction

In the light of rising computing paradigms such as Cloud computing ([1]), new value chains are emerging for outsourced hosting and execution of interactive multimedia applications. The latter have strict requirements on quality of service in order to operate effectively (e.g. latency, bandwidth and jitter for video streaming, response time to a request of the elearning server example presented later on). Actors in the value chain emerge where value can be added, e.g. at the infrastructure level through virtualized storage, networking and compute resources (Infrastructure-as-a-Service), at the application level through offering a specific software tool on a pay-per-use basis (Software-as-a-Service) and in-between these two levels, comes the possibility of Platform-as–a-Service (PaaS).

For the above value chain to support applications with real-time attributes, careful planning is required, so that neither under-provisioning (likely failure of the application to execute) nor massive over-provisioning (unnecessarily high costs) occur.

Furthermore, extensive use of techniques for incorporating applications with different characteristics in the infrastructure creates a burden with regard to the investigation of the application's behavior. Such techniques may include the use of virtualization, specialized scheduling and sharing of resources between different components. Conclusively, extensive data sets must be collected in an automated way so that conclusions regarding an application's behavior with varying resource allocations may be investigated.

In this paper, a process for gathering extensive datasets from applications inside the EU-funded project IRMOS ([11]) is described. This gathering incorporates state of the art components such as virtual machines (VMs) and real time schedulers, based on a variation of a number of parameters that are relevant to the investigated application and to the hardware configuration of the nodes of execution. Analysis and results regarding the effect of these parameters (such as changing scheduling granularity) to the application QoS levels are presented, in order to let the Service Provider (SP) use the fittest settings for the application under consideration. The resulting data sets will be made available to the general public for reusability purposes.

II. Related work

Similar work to the one presented in this paper is described in this section. In [2], DynBench is introduced, as a benchmark for distributed real time applications infrastructures. This creates dynamic conditions for the testing of the infrastructures. While promising, this framework is mainly oriented towards investigating the limits of the infrastructure and not towards understanding application behavior with different configurations.

In [3], VSched is presented, an EDF-based scheduling algorithm. In this work, an analysis is conducted on application performance with the scheduler in question, investigating the effect of scheduling decisions and concurrent virtual machines execution. The analysis is very thorough and interesting, however no framework is presented for obtaining the necessary data sets.

Manuscript received April 11, 2010. .This work is partially funded by the European Commission as part of the European IST 7th Framework Program through the project IRMOS under contract number 214777.

G. Kousiouris and D. Kyriazis are with the National Technical University of Athens 9, Heroon Polytechniou Str 15773 Athens, Greece, Tel.+302107722546;e-mail:gkousiou@mail.ntua.gr,dkyr@telecom.ntua.gr.

F. Checconi is with the Scuola Superiore S.Anna, Pisa, Italy. Mail: fabio@gandalf.sssup.it

A. Mazzetti is with Giunti Labs, Italy, Mail: a.mazzetti@giuntilabs.com.

Z. Zlatev and Juri Papay are with the IT Innovation Centre, University of Southampton, UK. Mail: {zdz, jp} @it-innovation.soton.ac.uk

T. Voith is with Alcatel Lucent Deutchland AG, mail: Thomas.Voith@alcatel-lucent.com

In [4], DIANE is presented for Grid-based user level scheduling with a focus on applications. However these applications are more centered around execution end time and not on real time interactivity.

A very interesting work is presented in [5] , where the users of a virtual machine are given the opportunity to increase through a simple interface their allocated CPU, based on their experience with the application. The cost of the increase is shown, so that the user may decide on the fly. While it is a very promising approach and would eliminate a vast number of issues with regard to application QoS levels, its main drawback is in cases of workflows. Inside a workflow, a degradation in performance may be due to a bottleneck on various nodes executing a part of it. The user will most likely be unaware of the location of the bottleneck, especially in cases of non experts.

Another interesting work with regard to real time scheduling and virtualization appears in [12]. In this case, the schedulability of concurrent virtual machines is investigated, in relation to the application deadlines met. Our work differs from this due to the fact that in this paper one of the major goals is to investigate application behavior with regard to changing scheduler assignments. The same applies for the work presented in [13], which compares different scheduling algorithms. The framework presented here is more application centric but can also be used for comparison purposes of the effect of these schedulers on application performance.

The remainder of the paper is structured as follows. In Section III, the role of the proposed framework for application sampling and analysis inside the IRMOS Framework is presented, along with details regarding the parameters of concern. In Section IV, the testbed used for the automatic collection of data is described, while in Section V the description of the process is presented. Finally, we present an analysis on the created data set, with a focus on the effect of the altered parameters on the application QoS level (Section VI) and conclusions (Section VII).

III. ROLE OF SAMPLING IN IRMOS

The major goal of IRMOS (Interactive Real-Time Applications on Service Oriented Infrastructures) is to enable the utilization of distributed infrastructures such as Service Oriented Infrastructures (SOIs) for interactive soft real time applications. In order for this to be accomplished, the most significant challenge is to offer guaranteed levels of QoS to the applications running inside the framework. However, these software components may be in many cases proprietary. Acquisition of sufficient information in order to deduce conclusions for their behavior can only be achieved through a macroscopic view. What is more, it is assumed that the applications are not written specifically for operation as IRMOS services, but rather, software components already in general use wrapped up as SaaS applications. As a consequence the actual internal operation of the application will be very difficult to be ascertained and used for the purposes of performance modeling. One way to collect this type of information is through executing the application for a variety of different parameters and determine their effect on the QoS output.. Through this information the IRMOS provider will be able to have an idea regarding what kind of resources should be allocated in order to meet the QoS levels requested by the client. Processing of these data for interpolation may be performed in a variety of ways (analytical modeling, statistical analysis, queueing theory, artificial intelligence etc.) however we consider this part out of scope for this paper.

For real time applications, the basic aim is to provide QoS guarantees. These may be either extremely strict, with no possibility to fall below the specified levels (hard real time constraints) or more relaxed, allowing for a predefined percentage of the QoS output to be above the wanted levels (soft real time constraints) ([8]) . In IRMOS, the second case is considered. So, what is critical, is to have a probabilistic approach that covers the needed levels.

The data sets needed for the creation of these probability distributions are obtained by general experimentation and sampling activities that are described in this work. Application runs can be performed for a series of workloads, with different application setup, on a variety of hardware configurations. More details regarding the modelling approach followed in the project can be found in [9].

A. Sampling Parameters

The parameters for which these sampling tests will be conducted depend on both the hardware on which the application is executed and the application parameters that will be toggled during real life executions.

For the hardware parameters, different CPU percentage assignments can be given with varying granularity. The granularity concerns the time period in which this percentage is assigned and can vary from a few milliseconds to seconds typically. This affects the performance of an application (for the same percentages of CPU allocation) in many ways since different needs must be met in each occasion. For example, for interactive real time cases, the application must be able to be activated in specific time intervals in order to give the user the notion of interactivity. This period may be different from case to case. However frequent task switching in the CPU results in increased overheads for the switching process and the restoration of each task's status. So a trade-off must be achieved between the two cases. On the other hand, on applications such as scientific simulations this effect may be different. These applications are typically time consuming and have no need for interactivity. Thus, the larger this granularity is, the better the application behavior in terms of overall execution time will be, since with reduced task switching, cache utilization will be improved.

Other parameters have to do with the workload produced by the application. Different executions may produce different amount of work for the processor to handle. The effect of these factors must also be investigated in the context of service oriented infrastructures. For example, in a server based application, this parameter is determined by the number of users that produce requests towards the server. The higher this value is, the more requests are generated and the more strenuous to the resources the application will be.

IV. SAMPLING TEST-BED

The sampling test-bed consists of a number of necessary components in order to alter the aforementioned parameters. These include the Virtual Machine inside which the application resides and is executable on all nodes of the infrastructure. The second necessary component is the real time scheduler used, that allocates the CPU share to the VM process and alters the granularity of this assignment. Finally, the application that is installed inside the VM is needed in addition to an external simulator. The latter is used in order to create application workload. More details regarding the role of each component are given in the following sections.

A. Virtualization Approach

Expanding network connectivity and the growing bulk of data demands for larger infrastructures, which are able to react dynamically on computing, networking and storage needs. The concept, which provides “on demand” services by sharing infrastructure resources and maintaining reliability and scalability, is associated with the term “cloud”. On the 32th IETF meeting in 1995 ([10]) the term cloud has been already used for the telecommunication infrastructure – now known as telco cloud – dealing with IP routing over large “shared media” networks. Sharing the computing resources over time (perceived already by John McCarthy in 1961) is experiencing now a renaissance due to the virtualization technologies. Virtualization of computing resources allows running multiple operating systems time-shared on a single computer in so called virtual machines. The independence of the virtual machine from the real hardware allows it to provide the computing as an infrastructure service on demand on any real host with enough free computing power. The virtual machine needs to be light-weight for movement and for instant availability. The three main pillars computing, data storage and networking can be provided as a service on demand as long as there are some guarantees associated with it. The cloud infrastructure service - Infrastructure as a Service (IaaS) means that the infrastructure can be utilized as a service without expertise or control over the technology infrastructure ensuring certain guarantees. The guarantees for computing belong to virtual machines experiencing certain CPU time over the complete service time. This is an inevitable prerequisite for enabling a real-time application inside a virtual machine with certain CPU time guarantees. A real-time capable OS of the virtual machine makes it possible to run real-time tasks inside.
Inside the IRMOS framework, the Kernel-based Virtual Machine tool is used. For each application a VM is created that covers its functional requirements (OS, specific internal tools etc.) and which then can be executed on all nodes of a distributed infrastructure. Through the use of VMs, other parameters may easily be altered such as number of underlying cores used, memory size, CPU model etc.

B. Host Scheduler Description

In order to provide scheduling guarantees to the VMUs, we used a hybrid deadline/priority (HDP) real-time scheduler ([6]) developed within the IRMOS consortium for the Linux kernel. This scheduler provides temporal isolation among multiple possibly complex software components, such as entire VMUs. It uses a variation of the Constant Bandwidth Server (CBS) algorithm, ([7]) based on Earliest Deadline First, for ensuring that each group of processes/threads is scheduled for Q time units (the budget) every interval of P time units (the period). The CBS algorithm has been extended for supporting multi-core (and multi-processor) platforms, achieving a partitioned scheduler where the set of tasks belonging to each group may migrate across the associated CBS scheduler instances across processors, according to the usual load-balancing heuristic of Linux. Furthermore, whenever each (partition of a) reservation is scheduled on each core, the associated tasks are scheduled according to their real-time priorities.
The scheduler exhibits an interface towards user-space applications based on the cgroups framework, which allows for configuration of kernel-level parameters by means of a filesystem-based interface. This interface has been wrapped within a Python API, in order to make the real-time scheduling services accessible from within the IRMOS platform. The parameters that are exposed by the scheduler are the C and D values, where C is the amount of computing time assigned to the VM every D interval.

C. Application Description and Preparation

The application under investigation is an eLearning mobile instant content delivery, in which real-time requirements are combined with service oriented architecture. In this scenario a user can receive on his/her mobile phone some eLearning contents relevant to the position where she is (e.g. walking near to historical monument). It consists of a Tomcat based e-learning application that incorporates a MySQL database (Figure 2). The application is able to receive queries with GPS data from a client, search internally in the database and respond with an elearning object identifier that corresponds to the provided GPS coordinates (Figure 1). It is provided as a war file and installed inside the VM. The real-time requirement is mainly the response time in each request and depends on the number of concurrent users and the size of the downloaded contents.

Furthermore, it must provide a way for the sampling framework to gather the reported data with regard to the values of interest. In the examined application, this transition of information was implemented with two potential ways. The first one was an XML report available through a URL. The sampling framework polled this URL with a given frequency and the XML report was stored and processed afterwards. The second option was for the application to store on its own the reports inside a MySQL database, from which the sampling framework could retrieve them.

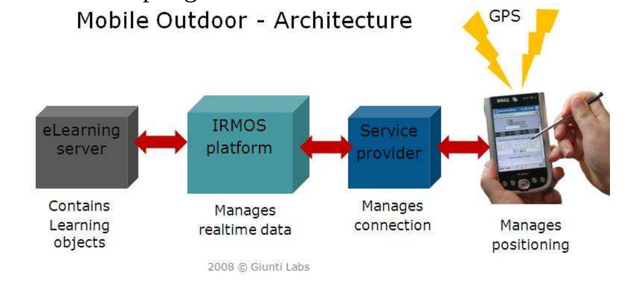


Figure 1

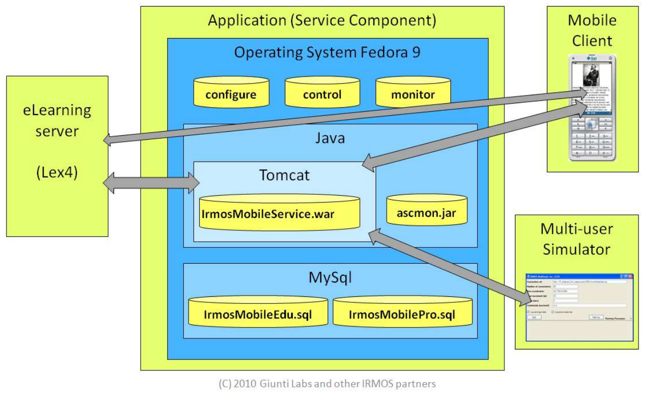


Figure 2: Application Design

D. Application Client Simulation Description s

In order to simulate application parameters, a client simulator is also necessary. This simulator can toggle the number of users performing queries on the server, thus varying the server load. By having different server loads and different hardware configurations we can have an analysis of their effect on the expected QoS output (in this case the response time of the server to the users).

One significant advantage of the test-bed is that the components described are decoupled from one another. This makes it flexible, so that these components (like schedulers, different virtualization tools or applications) can be replaced with different versions, thus making comparisons between them easy.

V. Collection and Processing Framework

In order for the collection of the samples to be conducted as automatically as possible, a number of actions have been implemented. First of all, the application client simulator is started, with a fixed number of users, whose created traffic is simulated. Afterwards a Java-based program resides on the physical host level of the infrastructure. This code is responsible for retrieving the reported monitoring data from the application. This can be done with two ways. The first case is to call the URL provided by the application and described above in order to collect the XML reports produced by the latter. The reports from every sample of one configuration are appended in a single XML file, whose name is indicative of the scheduling parameters used for the execution (C and D). Each sample is taken in a specific period, expressed through a parametric delay inserted between consecutive calls to the URL. This sampling frequency could be adjusted in case of periodic applications in order to obey to the Nyquist-Shannon theorem so that from the samples collected the entire distribution can be created. The second case is through the MySQL database, in which the reports from the application are timestamped and stored. For every configuration the start and end time are saved, and based on this information the application data that were stored during this interval are retrieved.

Furthermore, in the same code, a Java-system interface is implemented in order to be able to change the configuration of the scheduling parameters through the interface script described in Section IV.B. This way, consecutive configurations are tested automatically and their result in the QoS parameter of the application (response time) is recorded. During the time of each configuration, the previously examined retrieval framework takes the necessary measurements. The change in the scheduling parameters is two-fold, it involves the C/D value, which is the percentage of CPU assigned to the VMU (which can be considered in a way as a simulation of different CPU speeds) but also the granularity of D. This granularity is expected to affect application performance, as stated in Section III. Even with the same %CPU assignment, a very large value of D would result in a highly non-responsive service, especially for interactive applications, due to the large interval of deactivation. For other applications with no interactivity, e.g. scientific simulations, a large number of D could prove to be useful, due to reduction in task switches and better cache utilization.

In conclusion, the Java class is responsible for altering the C, D parameters (both ratio and absolute values), for connecting to the application interface (URL available XML reports or MySQL DB), for extracting the reported values, for processing them in order to produce the necessary statistics (in this case mean response time and standard deviation) and for creating the final output. This output is CSV files, that contain matrices that can directly be used by performance estimation methods. These include columns with the different number of users, different C, D parameters and the extracted statistics.

Finally, the number of users in the client simulator is changed and the process is initialized again. Due to the elastic form of the testbed, other parameters may also be investigated easily, such as the memory assignment to the VM, configured at the VM startup. In this particular application memory requirements were not extensive that is why it was decided not to investigate this parameter.

The structure of the sampling framework appears in Figure 3.

In order to extract the necessary information that is needed in the modeling approach followed inside IRMOS as described in Section III, the sampled response times of the eLearning server are gathered for each execution and statistical metrics are extracted. These can be used for the construction of the PDFs of the QoS output in consideration, for use in the next stages of modeling. The basic metrics that are extracted are the mean value and the standard deviation of the response times.

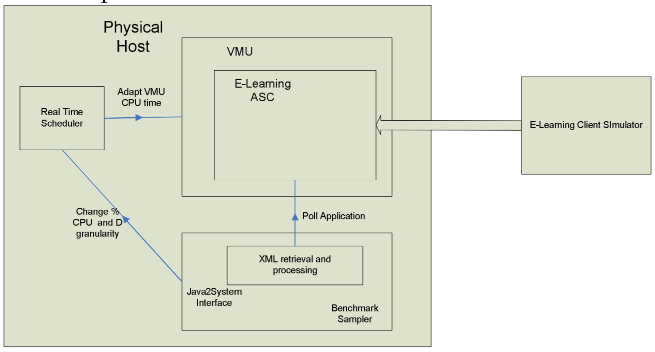


Figure 3: Sampling Framework

VI. Results

In this section, the results from the performed experiments are depicted. The range of values that were altered is:

- Number of Users: 30-150
- C/D (CPU share) : 20-100% with a step of 20
- D: 10000- 560000 (μsec) with a step of 50000

Measurements were taken and the gathered values for each configuration were collected. An average of 800 response times was collected for each different setup, in order to extract their mean and standard deviation values. An indicative set of these measurements is depicted in the following figures, from which useful conclusions can be drawn. Through the proposed framework, automated measurements could be performed for the 2 out of 3 parameters investigated (C/D and D). The number of users in the simulator had to be changed manually each time.

The effect of changing granularity on the deviaton of the response time values can be observed in Figure 4. This is expected since with high values of D, the service has long active and inactive periods. If the requests fall in the active interval, they will be satisfied quickly but if they fall in the inactive one then they will have to wait until this has finished. This effect decreases as allocated CPU share increases, since in these cases the CPU is almost dedicated to the application and whenever a request arrives it is served. The mean response time, as shown in Figure 5, seems not to be affected greatly given that the percentage of CPU assigned is the same.

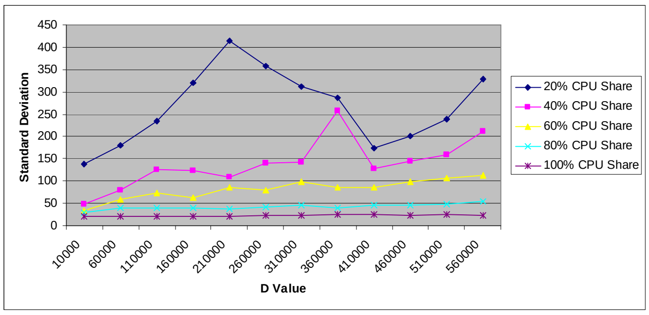


Figure 4: Standard Deviation with regard to changing D for 90 users

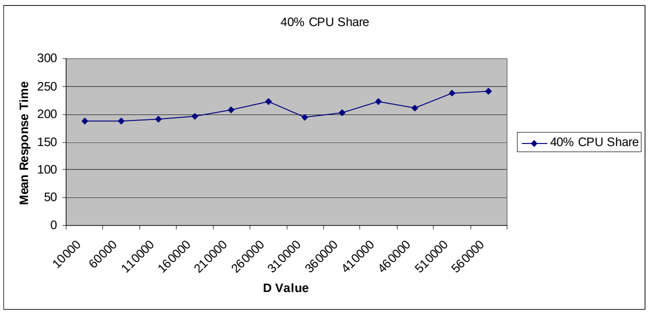


Figure 5: Mean value with regard to changing D for 70 users and 40% CPU share

In Figure 6, the comparison between the collected values of response times is shown for two different numbers of users. The difference especially in the maximum values of the distributions depicts the effect of the application workload in the response times.

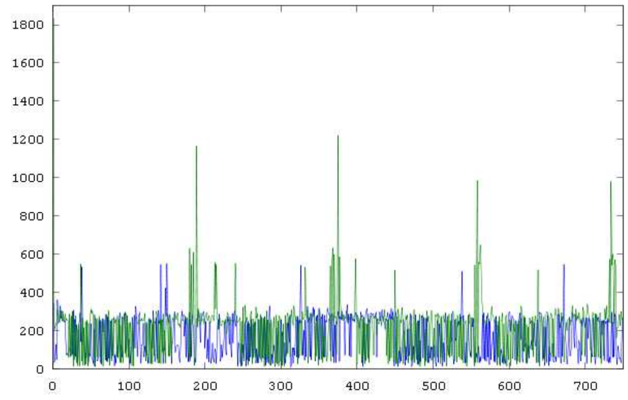


Figure 6: Comparison of different number of users (blue 30, green 50) for the same resources (40% of the CPU) and same D (x axis: samples, y axis: response time)

In Figure 8 all the different configurations are shown for two different numbers of users. In this case, each group of columns (the first high one followed by 4 lower ones) represents one D configuration for different percentages. The high bar is for low utilization and while the utilization increases the response time decreases. In the horizontal axis the different D configurations represent increasing D values.

From these measurements it seems interesting that the fittest granularity (D) selected depends also on the percentage of the CPU assigned to the application. In this occasion, for low percentages of utilization it is best to assign values near the middle of the investigated interval (10000-560000), as is depicted in Figures 7 and 8. For higher percentages of utilization, lower values of D are more beneficial for the response times of the application. Furthermore, from Figure 7 the effect of the increased CPU share allocation to the response time can be observed.

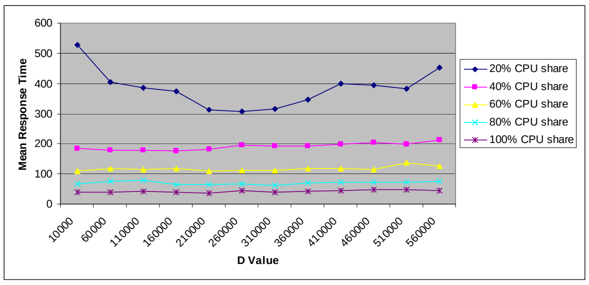


Figure 7: Mean Response Time for different D's and CPU shares for 110 users

The data set that was produced from the process described in this paper and that was the basis for the above figures will be made available to the community, following the initiative for reusable data sets.

VII. SUMMARY

In this paper, a sampling and analysis framework, used within the IRMOS project has been described. The aim of this framework is to easily gather extensive datasets regarding an e-Learning application and its real time requirements, in relation to characteristics such as the number of users of the application and the hardware allocation to it. This framework utilizes state of the art techniques in virtualization and real time scheduling, and the corresponding analysis of the results aids Service Providers in understanding the application's behavior. It is also flexible in order to be used in distributed infrastructures with no need for alterations for the deployment in a variety of nodes. This in turn can lead to enhanced allocation strategies. For the future, one interesting aspect to investigate would be the interference between co-scheduled VMs.

ACKNOWLEDGMENT

This research is partially funded by the European Commission as part of the European IST 7th Framework Program through the project IRMOS under contract number 214777.

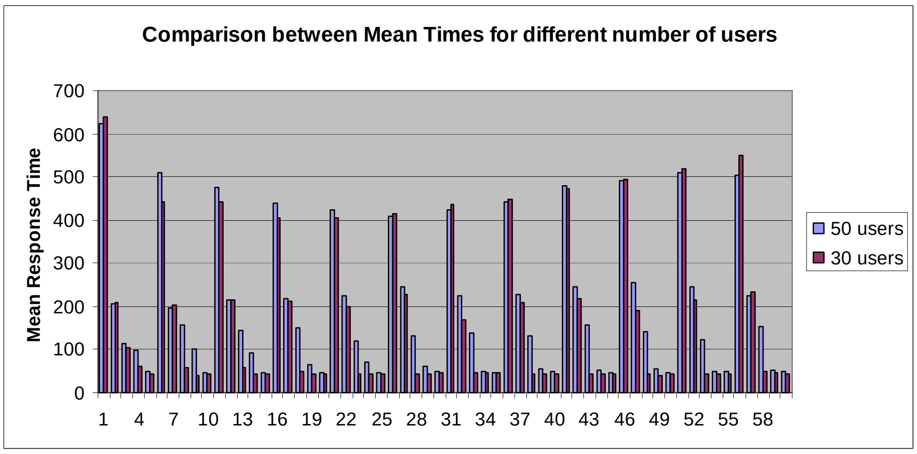


Figure 8

REFERENCES

[1] http://www.cloudcomputing.org/
[2] B. Shirazi, L. Welch, B. Ravindran, C. Cavanaugh, B. Yanamula, R. Brucks, E. Huh. DynBench: A Dynamic Benchmark Suite for Distributed Real-Time Systems. IPDPS Workshop on Embedded HPC Systems and Applications. S. Juan, Puerto Rico, 1999
[3] Lin, B., Dinda, P.: Vsched: Mixing batch and interactive virtual machines using periodic real-time scheduling. In: Proc. of the IEEE/ACM Conf. on Supercomputing, p. 8, Nov. 2005
[4] Germain, C., Loomis, C., Mo´scicki, J.T., Texier, R.: Scheduling for responsive Grids. J. Grid Computing 6(1), 15–27 (2008)
[5] Lin, B. and Dinda, P. A. 2006. Towards Scheduling Virtual Machines Based On Direct User Input. In Proceedings of the 2nd international Workshop on Virtualization Technology in Distributed Computing (November 17 - 17, 2006). Virtualization Technology in Distributed Computing. IEEE Computer Society, Washington, DC, 6. DOI= http://dx.doi.org/10.1109/VTDC.2006.15
[6] Fabio Checconi, Tommaso Cucinotta, Dario Faggioli, Giuseppe Lipari, "Hierarchical Multiprocessor CPU Reservations for the Linux Kernel," in Proceedings of the 5th International Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT 2009), Dublin, Ireland, June 2009
[7] Luca Abeni and Giorgio Buttazzo, "Integrating Multimedia Applications in Hard Real-Time Systems," in Proc. IEEE Real-Time Systems Symposium, Madrid, Spain, 1998
[8] .Liu, C. L. and Layland, J. W. 1973. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. J. ACM 20, 1 (Jan. 1973), 46-61. DOI= http://doi.acm.org/10.1145/321738.321743
[9] Matthew Addis, Zlatko Zlatev, Bill Mitchell, Mike Boniface, Modelling Interactive Real-time Applications on Service Oriented Infrastructures, Proceedings of 2009 NEM Summit, ISBN 978-3-00-028953-8
[10] http://www.ietf.org/proceedings/32/charters/rolc-charter.html
[11] http://www.irmosproject.eu/
[12] Cucinotta, T., Anastasi, G., Abeni, L.: Real-time virtual machines. In: Proceedings of the 29th IEEE Real-Time System Symposium (RTSS 2008) – Work in Progress Session, Barcelona (December 2008)
[13] B. Brandenburg and J. Anderson. A comparison of the M-PCP, D-PCP, and FMLP on LITMUSRT. In Proc. of the 12th International Conference On Principles Of Distributed Systems, pp. 105–124, 2008.

WATERS 2010 - Demo Session

This session features practical demonstrations of tools that have been considered interesting for the purposes of the workshop, with a special focus on applicability to real-time systems research.

A loadable task execution recorder for Linux

Mikael Åsberg, Johan Kraft and Thomas Nolte
MRTC/Mälardalen University
P.O. Box 883, SE-721 23,
Västerås, Sweden
{mikael.asberg,johan.kraft,thomas.nolte}@mdh.se

Shinpei Kato
The University of Tokyo
7-3-1 Hongo, Bunkyo-ku,
Tokyo 113-8656, Japan
shinpei@il.is.s.u-tokyo.ac.jp

***Abstract*—This paper presents a task recorder for Linux-based operating systems, in the form of a loadable kernel module. To the best of our knowledge, this is the first Linux task recorder which does not require kernel patches (kernel modifications). This complies with the requirements in the area of embedded systems where reliability and stability are important properties, hence, proven versions of Linux are therefore preferred.**

The implementation is based on the loadable real-time scheduler framework RESCH (REal-time SCHeduler). RESCH uses only exported Linux kernel primitives as a means for controlling scheduling. The disadvantage with this solution is that it can only detect scheduling events of the tasks being scheduled by RESCH itself, since it can not directly manipulate nor have knowledge of the tasks in the Linux task ready queue. In order to verify the correctness of the task recording, a comparison has been made with a second recorder, which uses a kernel patch. Our tests indicate that the new, RESCH-based, recorder gives identical results[1].

I. Introduction

The overall aim of our research is the development of hierarchical scheduling. Hierarchical scheduling has several advantages, stretching from enabling design time parallel development of system parts, simplifying integration, to runtime temporal partitioning and safe execution of tasks. Our previous work includes practical issues of this kind of scheduling [1] as well as the theoretical advantage [2] of this scheduling technique, in real-time systems. However, the hierarchical scheduling technique is rarely an integrated part of an operating system (except for, e.g., ARINC653 compliant operating systems that are commonly found in avionics applications). Indeed, there is a need to develop/implement new scheduling algorithms, such as hierarchical scheduling, in the area of real-time systems. Looking from a practical point of view, it is an advantage if hierarchical scheduling (and other scheduling techniques) can be implemented easily and efficiently without modifying the kernel. The latter makes it easier for both developers and users since there is no need to maintain/apply kernel patches (kernel modifications) every time the kernel is replaced/updated. Moreover, keeping the scheduler isolated in a kernel module, without modifying the kernel, simplifies debugging and potential certification of its correctness (component-based development advantages). We see that RESCH [3] is useful because it has the advantages mentioned.

RESCH is a scheduling framework intended to make life easier for scheduler developers in Linux based RT/GP OSs (Real-Time/General Purpose Operating System). A key motivation for using RESCH is that it does not need any kernel modifications, secondly, it makes scheduler development easier because it abstracts the complexity of scheduling aspects and presents a simple and easy scheduling interface to the user. However, while development of schedulers are simplified with this framework, it lacks support for debugging the schedulers. Our vision is to make the RESCH framework a complete scheduler development base. We want it to have all necessary tools for creating schedulers, and everything should be totally independent of kernel patches. This also has the advantage that it is easy to develop RESCH for other platforms, hence, making scheduler development platform independent. In order to comply with our vision, we want the integrated debugger in RESCH to be free of kernel patches as well.

With this paper, we present a task execution recorder, which is capable of debugging schedulers. The task recorder (we will refer to it as a recorder for the rest of the paper), is able to record the following scheduling events during run-time:

1) The time instance when a task is released (even though it might not start to execute).
2) The time instance when a task starts to execute.
3) When there is a task switch, the recorder distinguishes between preemption and non-preemption.
4) The time instance when a task finishes its execution.

The output from the recorder is a simple text-file containing task switch events. We have converted this file format to fit the trace visualization tool Tracealyzer[2].

A. System model

We assume fixed priority preemptive scheduling of periodic tasks, according to the periodic task model [4]. A task i is presumed to have the following parameters, $\langle T_i, WCET_i, D_i, pr_i\rangle$, where the period T_i represents the frequency in which the task is released for execution, $WCET_i$ is the worst case execution time of the task, the relative deadline D_i (within the period) is when the task must complete its execution (RESCH monitors this) and pr_i is the task priority (higher value represents higher priority). Also, all tasks are assumed to execute on the same core, i.e., single-core.

[1]The work in this paper is supported by the Swedish Foundation for Strategic Research (SSF), via the research programme PROGRESS.

[2]For more information about Tracealyzer, see http://www.tracealyzer.se/

B. RESCH

As mentioned previously, RESCH is a patch-free scheduling framework for Linux. It supports periodic tasks which can be scheduled in a fixed-priority preemptive manner. RESCH runs as a kernel module (in kernel space), giving both an interface to users in user space (e.g. a task specific interface like `rt_wait_for_period()`) as well as in kernel space. The kernel space API (Application Programming Interface) has the interface shown below:

1) **task_run_plugin( )**
2) **task_exit_plugin( )**
3) **job_release_plugin( )**
4) **job_complete_plugin( )**

These functions can be implemented by a **RESCH plugin** (Figure 1), i.e., a kernel module that has access to the RESCH kernel API. These functions are called in **RESCH core** at certain events. Functions 1) and 2) are executed every time a task registers/unregisters to RESCH. With register we mean that the task does a RESCH API call, transforming it to a **RESCH task**, which creates a RESCH TCB (Task Control Block) and puts it in the RESCH ready-queue etc. A RESCH TCB has, among other real-time specific data, a reference to its corresponding Linux task TCB (**task_struct**). The primitives 3) and 4) are called whenever a RESCH task is released for execution or when it has finished its execution. The plugins get these scheduling notifications and can thereby affect scheduling, trace tasks etc.

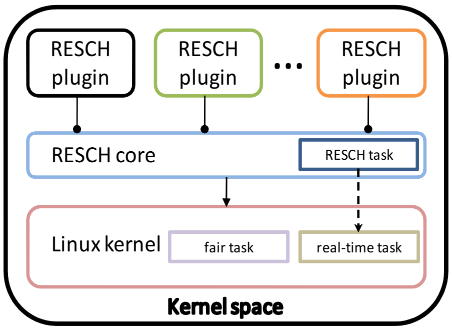


Fig. 1. RESCH framework

In Linux, since kernel version 2.6.23 (October of 2007), tasks can be either a **fair task** or a **real-time task**. The latter group has higher priority (0-99 where 0 is highest) than fair tasks (100-140). A task that registers to RESCH is automatically transformed to a real-time task. RESCH is responsible for releasing tasks, and tasks registered to RESCH must notify when they have finished their execution in the current period. In this way, RESCH can control the scheduling. RESCH uses an absolute-time clock, i.e., it does not wrap around. Also, release times are stored as absolute values, so release patterns are exact.
The cost of having a patch-free solution is that RESCH can only see scheduling events related to its registered tasks, i.e., higher priority real-time tasks, which are not registered in RESCH, can thereby interfere with RESCH tasks without the RESCH core detecting it. A simple solution to this problem is to schedule all real-time tasks with RESCH.

C. Task-switch hook patch

Our previous work [5] includes an implementation of a **task_switch_hook** (Figure 3), residing in a kernel module, which is called by the Linux scheduler at every scheduler tick. This solution requires modification of two code-lines in two separate kernel source files (*sched_rt.c* and *sched_fair.c*). The modification of file *sched_rt.c* is illustrated in Figure 3 (a similar change is done in *sched_fair.c*). Linux has (since kernel version 2.6.23) two scheduling classes, namely the *fair* and the *real-time* scheduling classes. When a new task should be released, the Linux scheduler iterates through its scheduling classes (first the *real-time* class, secondly the *fair* class) in order to find the next task to release. This is shown in Figure 2.

```
class = sched_class_highest;
for (; ; ) {
    p = class->pick_next_task(rq);
    if (p)
        return p;
    class = class->next;
}
```

Fig. 2. Kernel function: **pick_next_task**

The modification (Figure 3) makes it possible to re-direct a scheduling class' function **pick_next_task** to a user defined function (i.e., our function **task_switch_hook**), in a kernel module. This function (hook) can be inserted and removed during runtime.

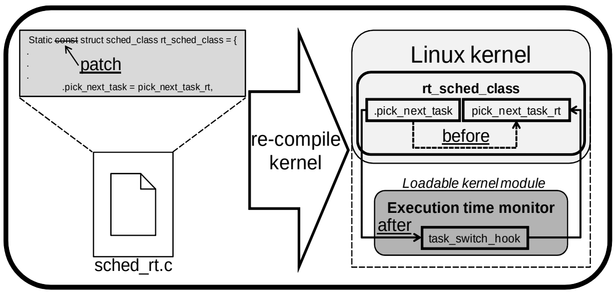


Fig. 3. Hook patch

In this paper our overall goal is to implement a patch-free task execution recorder in Linux for debugging purposes, i.e., which can be useful for a scheduler or application developer. Our solution enables debugging on any Linux-based RT/GP OS, as long as the Linux interface is not changed, since RESCH and the recorder are both loadable kernel modules that calls the Linux kernel functions. Hence, our recorder is more general than patched solutions since it is difficult to port these between different platforms.

The main contributions of this paper are:

1) We have implemented a (patch-free) task recorder with the use of RESCH, which enables debugging at task level, in Linux based RT/GP OS.

2) We have evaluated our solution by implementing yet another (patched) recorder, using the technique presented in [5], and compared the results from the two recorders.

The outline of this paper is as follows: in Section II we describe the two recorder implementations. Section III compares the trace result from the two recorders. Section IV presents related work, and finally, Section V concludes.

II. Implementation

The following section presents a recorder implementation based on RESCH, and a second implementation based on a hook patch [5].

A. RESCH plugin recorder

The recorder is implemented as a plugin (see Figure 1) in RESCH. Although, it could also become an integrated part of RESCH core at a later stage.

It is important to note that in order for the recording to be correct with this plugin, no higher priority real-time tasks (that are not registered by RESCH) are allowed to run. Also, the current recorder implementation does not support multi-core, hence, *load balancing* must be disabled (a function in Linux that migrates tasks to other CPUs based on load). Support for multi-core is possible, but without *load balancing*. The reason is that RESCH cannot detect task migrations made by the Linux scheduler.

Figure 4 shows the necessary data needed to store a scheduling event (i.e. task switch). The member **rid** is the index to the RESCH task TCB. We use this identifier because is has a smaller range than the Linux task id (PID), and thereby require less memory. The **timestamp** is stored in micro-seconds, i.e., our recorder can record approximately 35 minutes (assuming we have 32 bits) since we use one bit for storing the preemption flag (informing whether there has been a preemption or not).

```
struct task_switch_event {
  char next_rid; // rid (0 − 64) is the RESCH task id.
  char prev_rid;
  unsigned int timestamp; // Bit nr 31 hold preempt. flag.
};
```

Fig. 4. Event structure

`task_run_plugin`, line (1) Figure 5, is called every time a task registers to RESCH. Since the current version of our recorder does not support multi-core, we migrate all tasks to one CPU (CPU #0 in this case).

Figure 5 show parts of the recorder implementation (in simplified form) in RESCH. Line (4) and (18) (Figure 5) are called by the RESCH core at every task release and completion. In this way, not only can we record task switches, but also detect when a task is released and also show this information graphically in the Tracealyzer.

```
1: void task_run_plugin(resch_task_t *rt) {
2:    migrate_task(rt, 0); // Migrate all tasks to CPU 0.
3: }
4: void job_release_plugin(resch_task_t *rt) {
5:    resch_task_t *curr;
6:    int timestamp;
7:    timestamp = linux_timestamp_microsec( );
8:    curr = active_highest_prio_task(rt->cpu_id);
9:    if(curr == NULL) {
10:     store_event(IDLE, rt, NO_PREEMPT, timestamp);
11:     return;
12:   }
13:   if(rt->prio > curr->prio)
14:     store_event(curr, rt, PREEMPT, timestamp);
15:   else
16:     store_event(curr, rt, NO_PREEMPT, timestamp);
17: }
18: void job_complete_plugin(resch_task_t *rt) {
19:   resch_task_t *next;
20:   int timestamp;
21:   timestamp = linux_timestamp_microsec( );
22:   next = active_highest_prio_task(rt->cpu_id);
23:   if(next == NULL)
24:     store_event(rt, IDLE, NO_PREEMPT, timestamp);
25:   else
26:     store_event(rt, next, NO_PREEMPT, timestamp);
27: }
28: void store_event(resch_task_t *prev, resch_task_t *next,
29:                  char preempt, unsigned int timestamp) {
```

Fig. 5. Recorder implementation

B. Hook patch recorder

As mentioned previously, our patched recorder is based on a task-switch hook implementation [5]. This implementation consists of two hooks. One hook is executed when Linux calls the scheduling class *real-time*, the other one when *fair* class is called. The two hooks are never called in the same scheduler tick, only one of them (depending on if there are any real-time tasks eligible to execute). We use the similar data-structures in this implementation as the one presented in section II-A. A difference between the two approaches is that the hook patch implementation only detects a switch between tasks, i.e., it cannot know when a task is released (for which the RESCH implementation can). This will differentiate the trace results a bit.

III. Evaluation

We have tested our RESCH plugin recorder by running it in parallel with the hook patch implementation, i.e., the two recorders recorded the same trace at the same time. The recorder we compare with [5] was chosen because of its simplicity (easy to install, load/unload, modify source code etc.), small amount of source code and the fact that it records correctly since its hook is placed at the point where the Linux scheduler does the task context switches. We ran the task set in Table I on an Intel Pentium Dual-Core (E5300 2,6GHz) platform, equipped with a Linux kernel version 2.6.31.9, running with *load balancing* disabled. The recorded tasks ran on the same core, i.e., all tasks were migrated to CPU #0 at initialization phase. The trace from both implementations are visualized in Figure 9 and 10 with the Tracealyzer application. The tasks were scheduled by the RESCH core scheduler (i.e., we used no plugin scheduler) according to the parameters in Table I.

Name	T	$WCET$	D	pr
rt_task1	4	1	4	4
rt_task2	5	1	5	3
rt_task3	8	2	8	2
rt_task4	9	2	9	1

TABLE I
TASK SET USED IN THE EXPERIMENTS

Note that the absolute time-line in Figure 9 is 1 second behind Figure 10, but the relative time should match eachother since they were executed at the same time. The task ID:s in Figure 6 are the new modified PIDs (which are needed in order to reference RESCH TCBs from native Linux TCBs) and therefore different than the ones in Figure 7, which are the native PIDs.

If the user marks a task fragment, Tracealyzer will display a red box (e.g. task **rt_task4** in Figure 9 and 10) around the task instance, if it can detect the instance (which is not the case in Figure 10). In Figure 9 though, the Tracealyzer can separate between task instances, since the plugin recorder records task releases, e.g. line 20-21 in Figure 7, which represents the fourth fragment of **rt_task4** in Figure 9.

Figure 6 and 7 shows the data recorded by both recorders. The data is represented in the format: prev: <id1> <name1> next: <id2> <name2> <t-stamp> <preempt>, where **id1** and **name1** represents the task id and name of the task that is finishing (possibly preempted), **id2** and **name2** represents the task id and name of the task that is starting to execute, **t-stamp** is the timestamp in absolute time when this event has occurred and the flag **preempt** is set to 1 or 0 depending on if there is a preemption (1=preemption). The recorded data (Figure 6 and 7) corresponds to the graphical representation (Figure 9 and 10) from the start of the graphs (4.350.156 respectively 3.350.160) to the marked time-lines (14.354.148 respectively 13.354.149). The time in the graphs are represented in the format **second.milli-second.micro-second**. The difference in time (1 second) is due to that both recorders record absolute timestamps (and the tool visualizes in absolute time) and one recorder was started approximately 1 second before the other one.

```
1:  prev: 0 idle next: 32769 rt_task1 3350160 1
2:  prev: 32769 rt_task1 next: 32770 rt_task2 4266631 0
3:  prev: 32770 rt_task2 next: 32771 rt_task3 5183029 0
4:  prev: 32771 rt_task3 next: 32772 rt_task4 7038580 0
5:  prev: 32772 rt_task4 next: 32769 rt_task1 7350149 1
6:  prev: 32769 rt_task1 next: 32772 rt_task4 8266565 0
7:  prev: 32772 rt_task4 next: 32770 rt_task2 8354148 1
8:  prev: 32770 rt_task2 next: 32772 rt_task4 9270908 0
9:  prev: 32772 rt_task4 next: 0 idle 10726636 0
10: prev: 0 idle next: 32769 rt_task1 11350157 1
11: prev: 32769 rt_task1 next: 32771 rt_task3 12266246 0
12: prev: 32771 rt_task3 next: 32770 rt_task2 13354149 1
```

Fig. 6. Recorded data (Hook patch)

```
1:  prev: 0 idle next: 3900 rt_task1 4350156 1
2:  prev: 3900 rt_task1 next: 3901 rt_task2 4354149 1
3:  prev: 3901 rt_task2 next: 3900 rt_task1 4354159 0
4:  prev: 3900 rt_task1 next: 3902 rt_task3 4354150 1
5:  prev: 3902 rt_task3 next: 3900 rt_task1 4354160 0
6:  prev: 3900 rt_task1 next: 3903 rt_task4 4354151 1
7:  prev: 3903 rt_task4 next: 3900 rt_task1 4354161 0
8:  prev: 3900 rt_task1 next: 3901 rt_task2 5266627 0
9:  prev: 3901 rt_task2 next: 3902 rt_task3 6183025 0
10: prev: 3902 rt_task3 next: 3903 rt_task4 8038576 0
11: prev: 3903 rt_task4 next: 3900 rt_task1 8350148 1
12: prev: 3900 rt_task1 next: 3903 rt_task4 9266561 0
13: prev: 3903 rt_task4 next: 3901 rt_task2 9354147 1
14: prev: 3901 rt_task2 next: 3903 rt_task4 10270905 0
15: prev: 3903 rt_task4 next: 0 idle 11726633 0
16: prev: 0 idle next: 3900 rt_task1 12350153 1
17: prev: 3900 rt_task1 next: 3902 rt_task3 12354149 1
18: prev: 3902 rt_task3 next: 3900 rt_task1 12354159 0
19: prev: 3900 rt_task1 next: 3902 rt_task3 13266243 0
20: prev: 3902 rt_task3 next: 3903 rt_task4 13354147 1
21: prev: 3903 rt_task4 next: 3902 rt_task3 13354157 0
22: prev: 3902 rt_task3 next: 3901 rt_task2 14354148 1
```

Fig. 7. Recorded data (RESCH plugin)

The difference between the two traces is that the RESCH plugin recorder records task releases by recording a fake preemption and running the released task for 10 micro-seconds, e.g., lines 1-7 in Figure 7, though this is not the case (its just for visualization). Although, this enables the Tracealyzer to calculate and show the response time of a task. Other than that the preemption depth differs in the two traces (due to that the two recorders record preemption differently), the traces are almost identical (the recorded time points may differ a few micro-seconds at most). Figure 8 shows the RESCH plugin trace, visualized with the tool Grasp [6]. The tasks vertical positions are ordered by priority with the lowest priority at the top (including the idle task) and the time is scaled down 100000 times. The figure clearly shows the task frequency.

IV. Related work

The idea of our solution is based on the replay debugging approach [7], which records system events online and replays them offline. In later work [8], the replay debugging has been extended to be compiler- and OS-independent. While the replay debugging works with off-the-shelf compilers for application-level debugging, our solution is self-contained software using Tracealyzer [9] for OS-level debugging, and it is primarily focused on real-time scheduler debugging.

The SCHED_DEADLINE project [10], which is in charge of the EDF scheduler implementation for Linux, has used the sched_switch tracer provided by the Ftrace toolkit [11] to output the records of context switches. The output logs are then converted to the VCD (Value Change Dump) format so that GtkWave can visualize the task execution traces. The trace can of course be converted to the Tracealyzer or Grasp [6] format. Given that Ftrace is supported by the Linux community, it is reasonable to use this toolkit to trace task executions for kernel debugging, but it is dedicated to the Linux kernel, so it is not necessarily suitable for real-time scheduler debugging. For instance, sched_switch does not catch job releases, however,

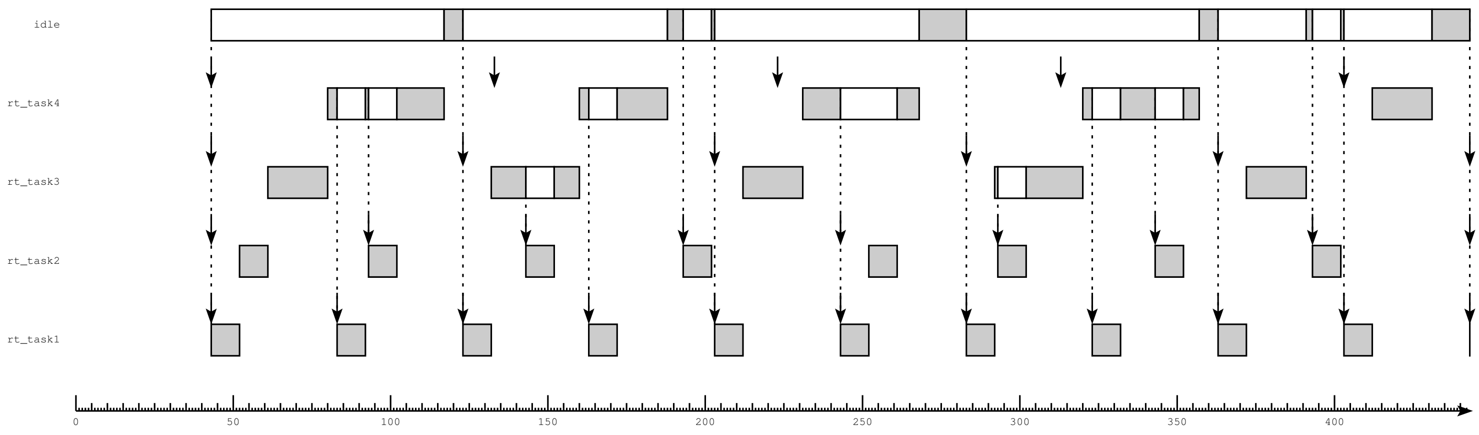


Fig. 8. RESCH plugin trace visualized with Grasp [6]

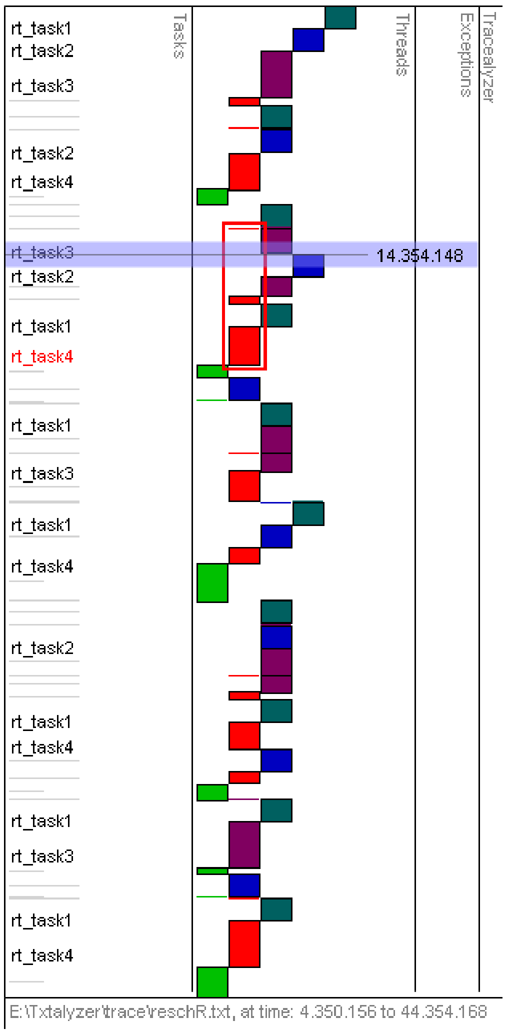


Fig. 9. Example trace with RESCH plugin

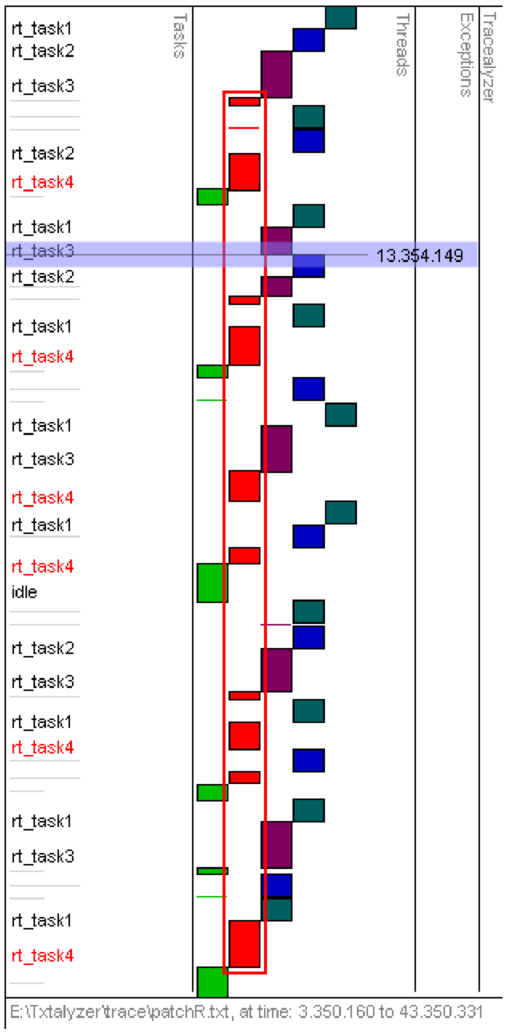


Fig. 10. Example trace with hook patch

context switches are precisely traced and it can distinguish between task completions and task preemptions. Our solution is more flexible and integrated in that it is available not only for the Linux kernel but also for other OSs, once the RESCH framework is ported.

DTrace [12], SystemTrap [13], LTT [14], and LTTng [15] are advanced tools for OS debugging. They are oriented for tracing entire kernel events, so it is required that the developers, in a high degree, understand how to use them. Meanwhile, our solution is more simplified by focusing on real-time scheduler debugging, and it is very easy to use in practice.

Real-Time Application Interface for Linux (RTAI) [16] is a collection of loadable kernel modules and a kernel patch which together provides a rich real-time API to the user. It gives the possibility to add/delete hooks for every task-start, task-switch and task-delete. These hooks give the possibility to monitor task execution in a detailed level.

V. Conclusion

We have implemented a task execution recorder in a stock Linux kernel, without applying kernel patches, with the use of the loadable real-time scheduler framework RESCH. The recorder is able to record task releases (with or without preemption) and task switches. The recorded data is later converted, offline, to a format suitable for the visualization tool Tracealyzer. In this way, the trace can be visualized graphically. The assumptions made, in order for the tracing to be correct, are that there should not exist any (unregistered RESCH) tasks that have higher priority than the (RESCH) tasks to be recorded, and that *load balancing* is disabled. Our results indicate that our recorder does a correct trace, by comparing the results with a trace made by a patched recorder. The second (patched) recorder is assumed to trace correct, since it is called by the Linux scheduler at every scheduler tick. The two recorders were executed in parallel, i.e., they recorded the same trace. Hence, an exact comparison is possible since the execution time of the tasks will be the same in both cases, and will therefore not affect the comparsion results. We showed that our recorder got the same trace result as the patched solution (with only a few micro-seconds of difference).

As future work we will continue with evolving the RESCH framework. This includes the development of new scheduler plugins, such as hierarchical scheduling (for both uni- and multi-core), adjust our recorder to fit with multi-core and implement RESCH for other platforms. In this way, plugin schedulers, recorders etc. can be moved to other platforms (supported by RESCH) without modification. We will also explore the possibilities of doing visualizations of the trace during run-time, rather than offline as in this paper.

References

[1] M. Behnam, T. Nolte, I. Shin, M. Åsberg, and R. J. Bril, "Towards hierarchical scheduling on top of VxWorks," in *Proc. of the OSPERT workshop*, 2008.

[2] M. Åsberg, M. Behnam, F. Nemati, and T. Nolte, "Towards Hierarchical Scheduling in AUTOSAR," in *Proc. of the ETFA conference*, 2009. [Online]. Available: http://www.mrtc.mdh.se/index.php?choice=publications&id=1793

[3] S. Kato, R. Rajkumar, and Y. Ishikawa, "A Loadable Real-Time Scheduler Suite for Multicore Platforms," Technical Report CMU-ECE-TR09-12, 2009. [Online]. Available: http://www.contrib.andrew.cmu.edu/~shinpei/papers/techrep09.pdf

[4] C. Liu and J. Layland, "Scheduling algorithms for multi-programming in a hard-real-time environment," *ACM*, vol. 20, no. 1, pp. 46–61, 1973.

[5] M. Åsberg, T. Nolte, C. M. O. Perez, and S. Kato, "Execution Time Monitoring in Linux," in *Proc. of the W.I.P. session in the ETFA conference*, 2009. [Online]. Available: http://www.mrtc.mdh.se/index.php?choice=publications&id=1792

[6] M. Holenderski, M. M. H. P. van den Heuvel, R. J. Bril, and J. J. Lukkien, "Grasp: Tracing, Visualizing and Measuring the Behavior of Real-Time Systems," in *International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems*, July 2010.

[7] H. Thane and H. Hansson, "Using Deterministic Replay for Debugging of Distributed Real Time Systems," in *Proc. of the ECRTS conference*, 2000, pp. 265–272.

[8] H. Thane, D. Sundmark, J. Huselius, and A. Pettersson, "Replay Debugging of Real-Time Systems Using Time Machines," in *Proc. of the IPDPS conference*, 2003, pp. 288–295.

[9] T. Maragria and B. Steffen, editors, "Leveraging Applications of Formal Methods," 1^{st} *International Symposium, ISoLA. Springer*, pp. 140–141, 2004.

[10] D. Faggioli and F. Checconi, "An EDF scheduling class for the Linux kernel," in *Proc. of the Real-Time Linux Workshop*, 2009.

[11] T. Bird, "Measuring Function Duration with Ftrace," in *Proc. of the Japan Linux Symposium*, 2009.

[12] B. Cantrill, M. Shapiro, and A. Leventhal, "Dynamic Instrumentation of Production Systems," in *Proc. of the USENIX conference*, 2004, pp. 15–28.

[13] V. Prasad, W. Colhen, F. Eigler, M. Hunt, J. Keniston, and B. Chen, "Locating System Problems Using Dynamic Instrumentation," in *Proc. of the Ottawa Linux Symposium*, 2005, pp. 49–64.

[14] K. Yaghmour and M. Dagenais, "Measuring and characterizing system behavior using kernel-level event logging," in *Proc. of the USENIX conference*, 2000, pp. 13–26.

[15] M. Desnoyers and M. Dagenais, "The LTTng Tracer: A low impact performance and behavior monitor of GNU/Linux," in *Proc. of the Ottawa Linux Symposium*, 2006, pp. 209–223.

[16] D. Beal, E. Bianchi, L. Dozio, S. Hughes, P. Mantegazza and S. Papacharalambous, "RTAI: Real Time Application Interface," *Linux Journal*, vol. 29, no. 10, 2000.

Grasp: Tracing, Visualizing and Measuring the Behavior of Real-Time Systems

Mike Holenderski, Martijn M.H.P. van den Heuvel, Reinder J. Bril and Johan J. Lukkien
Department of Mathematics and Computer Science
Technische Universiteit Eindhoven (TU/e)
Den Dolech 2, 5600 AZ Eindhoven, The Netherlands

Abstract—**Understanding and validating the timing behavior of real-time systems is not trivial. Many real-time operating systems and their development environments do not provide tracing support, and provide only limited visualization, measurements and analysis tools. This paper presents Grasp, a tool for tracing, visualizing and measuring the behavior of real-time systems. Grasp provides a simple plugin infrastructure for extending it with custom visualization and measurement methods. The functionality of Grasp is demonstrated based on experiences during the development of various real-time extensions for the commercially available μC/OS-II real-time operating system. All the tools presented in this paper are open source and freely available on the web[1].**

I. Introduction

A real-time system is usually comprised of a real-time application running on top of a real-time operating system. One such operating system is μC/OS-II [Labrosse, 1998]. It is maintained and supported by Micrium, and is applied in many application domains, e.g. avionics, automotive, medical and consumer electronics. Our choice of μC/OS-II is in line with our industrial and academic partners.

Based on the requirements posed by the real-time applications we are researching, we have set out on extending μC/OS-II with several real-time primitives. In [Holenderski et al., 2008] we presented an application of our industrial partner in the surveillance domain and pointed out a problem with their current system. We proposed a solution and identified several extensions required from the underlying real-time operating system. These include processor reservations based on deferrable servers, and support for resource sharing based on the Stack Resource Policy (SRP) [Baker, 1991].

A. Problem Description

During the development of these μC/OS-II extensions we needed to validate the behavior of the introduced primitives. Many commercial off-the-shelf real-time operating systems and their development environments, including μC/OS-II, do not support tracing, and provide only limited visualization and analysis support. Moreover, current visualization tools only allow to visualize single level scheduled systems. Finally, most commercial tools are not easily extensible.

[1]The work presented in this paper is supported in part by the European ITEA2-CANTATA project and the Dutch HTAS-VERIFIED project. The Grasp player together with two demo traces is available at http://www.win.tue.nl/~mholende/grasp

B. Contributions

In this paper we address the problem of tracing, visualizing and measuring the behavior of real-time systems and present Grasp, which is a set of tools addressing this problem. It comes with a powerful set of features out of the box, such as visualization of servers in hierarchical scheduling and buffer usage for tasks communicating via shared buffers. It also provides a simple infrastructure for extending it with custom visualization and measurement plugins. We used Grasp extensively during the development of several extensions of μC/OS-II. The target systems were executed in the cycle accurate OpenRISC simulator [OpenCores, 2009].

C. Outline

The remainder of this paper is structured as follows. In Section II we summarize the related work, followed by a description of our execution environment in Section III. Section IV is the main contribution of this paper. It presents Grasp, illustrated with examples from extending μC/OS-II with additional real-time primitives. Section V concludes the paper and outlines the future work.

II. Related work

In this section we outline the existing work related to the real-time operating system under consideration, followed by a discussion of the support for tracing, visualization, and measurements provided by existing tools.

A. μC/OS-II and its tools

Micrium provides the full μC/OS-II source code accompanied by an extensive documentation [Labrosse, 1998]. The μC/OS-II kernel provides preemptive multitasking, and the kernel size is configurable at compile time, e.g. services like mailboxes and semaphores can be disabled. It is well suited for proprietary extensions and experimentation.

A μC/OS-II application can enable a built-in statistics task, which collects information about the processor usage of all tasks in the system. Micrium also provides a powerful monitoring tool called μC/Probe, allowing to inspect the state of any variable, memory location, and I/O port in a μC/OS-II enabled application during runtime. However, there is no tracing support for μC/OS-II.

B. Tracing

There are basically two approaches to tracing: instrumentation and sampling [Mughal and Javed, 2008]. With instrumentation, code is inserted in key places of the system (such as the top of particular method calls). This code then records the events at runtime for later offline analysis. With sampling, the system remains unmodified and is instead analyzed periodically by a profiler during runtime, allowing inspection of metrics such as the amount of CPU time used by processes and/or functions. Grasp and all other tools presented in this section take the instrumentation approach, where a recorder component generates a trace file which later serves as input for the visualization application.

The format of the trace file has several implications. A standard textual file format (e.g. XML used by VET [McGavin et al., 2006]) can be used as input for other tools with relatively little effort. A binary file format (e.g. used by the Tracealyzer [Mughal and Javed, 2008]) results in smaller trace files, which can be important when tracing is a part of the target system after deployment in the field. A Grasp trace is a Tcl [Welch et al., 2003] script. It is less verbose than XML, but not as compact as a binary representation. However, its main advantage is the flexibility it offers for the Grasp player, as explained in Section IV.

C. Trace visualization

Traces contain huge amounts of data, which may be of no interest for a particular investigation. Several visualization tools therefore provide filtering mechanisms, which allow the user to display only those events he is interested in. Tracealyzer [Mughal and Javed, 2008] offers predefined filters which can be changed during the visualization, allowing to hide certain events, such as locking of a semaphore. VET [McGavin et al., 2006] provides a plugin mechanism allowing an expert user to implement custom filters, which then can then be reused by regular users. The Grasp player presented in this paper allows to filter trace events referring to certain tasks.

A trace can be visualized in different ways, e.g. one may want to show the task execution on a timeline, or how each task contributes to the current processor load. Tracealyzer provides a timeline and load view. VET provides a message sequence and class association diagrams, but also supports tracing of custom events, and an API which can be used to implement custom visualizations. This API, however is limited to a fixed event structure. The file format for Grasp traces allows to easily extend the trace with arbitrary custom events and visualizations.

To the best of our knowledge, all existing tracing tools can visualize only single level scheduling. Grasp on the other hand, also allows to visualize hierarchical systems by means of illustrating the budget consumption of servers. In case tasks communicate via shared buffers, Grasp can also provide insight into the contents of the buffers at any moment during the traced system execution.

D. Trace measurements

Tracealyzer measures the execution time, response time and number of fragments for each job and the corresponding worst case and average case values of all jobs of a task. Grasp measures the execution and response time of jobs and provides a summary of the average, best case and worst case times of all jobs of a task. It also allows to easily implement custom measurement tools, as illustrated in Section IV-D.

III. Simulation platform

To run our target systems we needed an execution environment supporting μC/OS-II. To avoid the inconveniences of running the target systems directly on hardware, we chose to run them inside the cycle-accurate OpenRISC simulator.

The OpenCores project [OpenCores, 2009] provides an open source platform architecture, including software development tools. The hardware architecture comprises a scalar processor and basic peripherals to provide basic functionality [Bolado et al., 2004]. The small and predictable processor architecture makes the OpenRISC processor suitable for real-time computing [Whitham and Audsley, 2006]. The accompanying Software Development Kit (SDK) is based on GNU tools. It includes a C/C++ compiler, linker, debugger and architectural simulator. The OpenRISC simulator allows simple code analysis and system performance evaluation. Recently, we created a port for μC/OS-II to the OpenRISC platform[2].

Unlike other μC/OS-II simulators, such as the Windows and Linux ports [uco, 2007], the OpenRISC port provides a cycle-accurate simulation: it is independent of the system load due to other applications on the host operating system and therefore provides predictable timing behavior for timed events.

It is important to note that the only interface to the simulator during runtime is via the standard input and standard output. In particular, there is no support for reading from or writing to files on the host operating system.

IV. Grasp

Grasp is composed of three entities, shown in Figure 1. The recorder is responsible for generating the trace of the target system. The generated trace file contains the raw data from a particular run, which is not comprehensible in its raw form. The player reads in a trace and displays it in an intuitive way.

Note that the recorder and the player are independent of each other, as long as the trace follows the predefined format. In this paper we demonstrate a Grasp recorder for μC/OS-II. Porting Grasp to other operating systems requires only implementing a Grasp recorder.

A. Grasp recorder

The Grasp recorder is implemented as a library providing functions to initialize the recorder, log events, and finalize the recorder. Calls to the event logging methods are inserted at several places inside the kernel to log common events, such as context switches and the arrival of periodic tasks. The

[2]A precompiled OpenRISC tool chain for Linux (Ubuntu 8.10) is available at http://www.win.tue.nl/~mholende/ucos

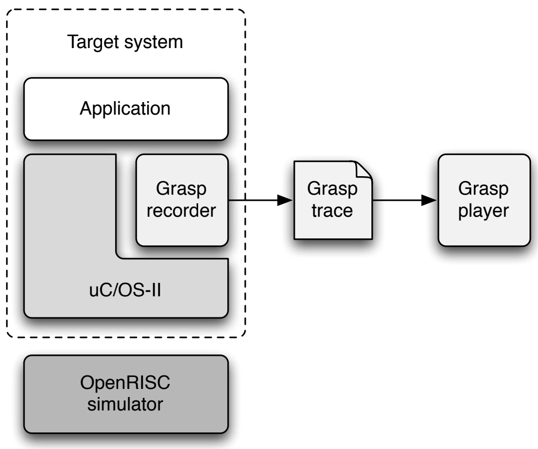


Fig. 1. The Grasp architecture

recorder also provides a function to log custom events, which the programmer may call inside his application.

To limit the interference with the target system, during run-time the μC/OS-II Grasp recorder stores the event information in a binary format in an array in the memory. Each event occupies 20 bytes of memory and takes 218 instruction cycles to log (on a 32-bit OpenRISC architecture).

At the end of a simulation the log array is traversed, a trace is generated and written to a file in text format. This way the I/O overhead associated with writing the trace to a file is postponed until the very end and limits the interference with the target system during a simulation run. Note that the OpenRISC simulator has no file system support, but does allow to print to the standard output via the `printf()` method. The recorder therefore prints the contents of the trace file to the standard output, which is then redirected to a file on the host operating system.

B. *Grasp trace*

The Grasp trace file is actually a Tcl [Welch et al., 2003] script. An excerpt from an example trace is shown in Figure 2.

```
1   ...
2   plot 50 taskArrived Task0x11da50
3   plot 50 jobPreempted Job0x11d948_1 \\
4           -target Job0x11da50_1
5   [Job Job0x11da50_2] initWithTask Task0x11da50 \\
6           -name "Task1 2"
7   plot 50 jobResumed Job0x11da50_2
8   plot 60 jobCompleted Job0x11da50_2
9   plot 60 jobPreempted Job0x11da50_2 \\
10          -target Job0x11d948_1
11  plot 60 jobResumed Job0x11d948_1
12  ...
```

Fig. 2. An excerpt from a trace file.

Each line in the trace is a Tcl command. A Tcl command has a very simple syntax: method name followed by a possibly empty list of arguments separated by spaces. Let us take a closer look at the first line in Figure 2, which indicates the arrival of task `Task0x11da50`. The meaning of this command is the following:

- `plot` is the method name responsible for handling this trace event. In Section IV-D we will see how this dispatching mechanism is used to implement plugins. The method determines the semantics of the following arguments. In this case, the `plot` method expects at least two arguments:
- `50` is the event time. Time is measured in ticks. In our simulations 1 tick corresponds to 1ms.
- `taskArrived` is the event kind.
- `Task0x11da50` is an additional argument. In this case it identifies the task which has arrived. In Grasp each trace object such as a task or a job has a unique identifier.

Figure 2 shows also several other events. For example at time 50 the job with id `Job0x11d948_1` is preempted by the job with id `Job0x11da50_1`. In the following sections we will describe other events supported by Grasp.

Every task or job referred to by an event needs to be created first. Line 5 in Figure 2 creates a job with id `Job0x11da50_2` for a task with id `Task0x11da50`. The optional parameter `-name` specifies a custom job name, referring to the second job of task `Task1`.

Note that since a trace is a Tcl script it may contain any Tcl code, in particular it may define its own methods, include loops, etc. While we do not exploit this feature in this paper, we have used it during the development of Grasp itself.

C. *Grasp player*

The Grasp player is written in the Tcl scripting language[1]. The job of the Grasp player is basically to provide an execution environment for the script inside a Grasp trace, by implementing all methods called in a trace file.

The Grasp player goes through the following stages:

1) Load the default methods, e.g. the `plot` method in Section IV-B.
2) Load any plugins, which define additional methods.
3) Read in and execute the trace script.
4) Do any post processing, e.g. export a postscript file.

Note that step 3 is a single Tcl command, but it is also the place where the main work happens and where the trace is actually visualized.

A handy feature of the Grasp player is the option to export the trace visualization to a postscript file and an option to print a legend. These are useful for automatically creating figures for research articles. An example of such a figure is shown in Figure 3, which visualizes the complete trace from Figure 2.

Shared resources: Grasp can visualize the synchronization of tasks in case they share resources.

Figure 3 demonstrates the behavior of three fixed priority scheduled tasks, with two of them (Task2 and Task3) sharing one logical resource according to the Stack Resource Policy [Baker, 1991]. The highest priority task has the lowest index number, i.e. Task1 has the highest priority. In this example, the execution of a critical section is visualized by

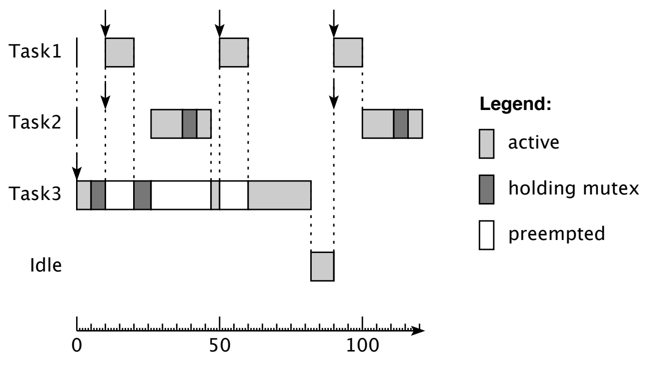


Fig. 3. Example trace visualization created by the Grasp player.

a dark section. Acquiring and releasing of mutexes is traced by the events `jobAcquiredMutex` and `jobReleasedMutex`, which are not visualized in the example.

Hierarchical scheduling: An interesting and unique feature of Grasp is the built in support for visualizing behavior of servers in a hierarchical real-time system. An example is shown in Figure 4.

There are four server events:

1) `serverReplenished` sets the capacity of a server.
2) `serverDepleted` creates a depleted message for a server.
3) `serverResumed` starts consuming a server's budget at a constant rate of 1 unit per time unit.
4) `serverPreempted` stops consuming a server's budget.

These four events are sufficient to visualize the behavior of most servers in the real-time literature. We have extended μC/OS-II with polling [Lehoczky et al., 1987], periodic idling [Davis and Burns, 2005] and deferrable servers [Strosnider et al., 1995].

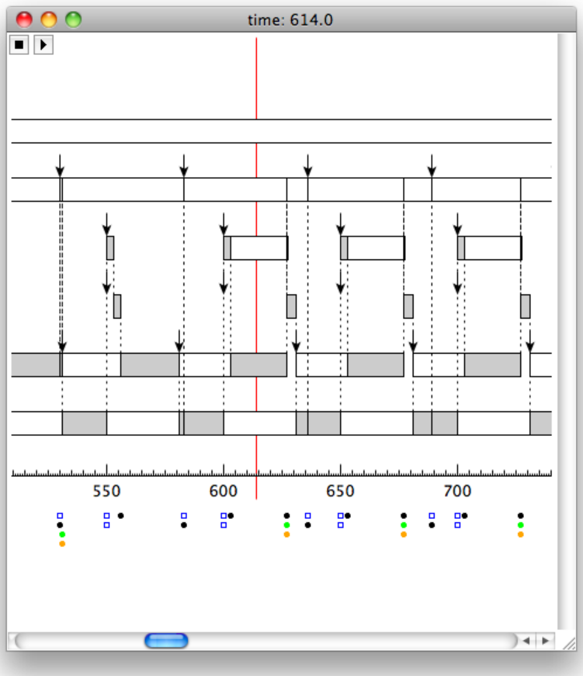
Figure 4 shows a Grasp player window after loading a trace file. The task execution is shown on top, with the server capacities illustrated underneath. In this particular example Task1 is assigned to the Deferrable Server, and Task2 is assigned to the Polling Server. The different shapes underneath the timeline indicate different events. For example, a triangle pointing upwards indicates a server replenishment and a square indicates the arrival of a periodic task. Clicking on a shape with the mouse reveals details about the event, e.g. the name of the server which is replenished.

The thin vertical line spanning across the tasks and servers is the *time marker*. It moves with the mouse cursor and indicates the current time in the visualization, also shown in the windows title bar.

During the development of the different servers Grasp provided useful insight into their behavior and speeded up the debugging process considerably.

D. Grasp plugins

Choosing the Tcl script format for a Grasp trace allows for a very simple interface for implementing plugins: a Grasp plugin is a set of methods which are called within a Grasp



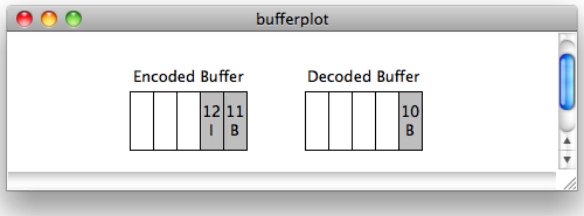


Fig. 5. Example of a trace visualization using the `BufferPlot` plugin.

trace. There are no restrictions on the syntax of the plugin methods, as long as they do not conflict with the default Grasp player methods.

To facilitate plugins which depend on the current time indicated by the time marker, Grasp provides an abstract event `<<TimeChanged>>`. A plugin can register for an abstract event using the Tcl `bind` command.

Plugins are loaded into a Grasp player by executing the player from the command line with the `-plugins` option followed by a list of paths to Tcl scripts implementing the plugins. Alternatively, the plugin scripts can be placed in the `plugins` subdirectory. All scripts residing in this directory are loaded automatically by the player.

In our recent work on mode changes in multimedia applications [Holenderski et al., 2009] we investigated an application comprised of a set of tasks communicating via shared buffers. We used Grasp to gain insight into the behavior of buffers and to measure the mode change latencies. We have implemented two plugins for this purpose.

The `BufferPlot` plugin defines four new events: `push`, `pop`, `insert` and `drop`. `BufferPlot` is implemented as a XOTcl class, which is an object oriented extension for Tcl.

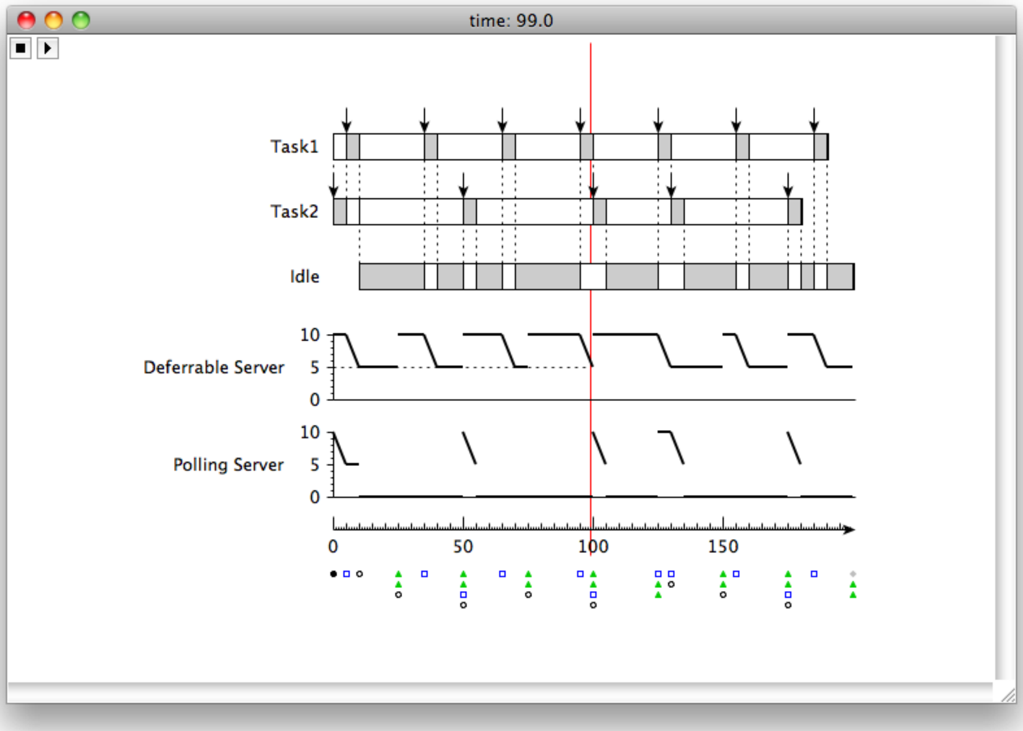


Fig. 4. Example of a trace visualization for hierarchical scheduling. The Task1 and Task2 tasks are assigned to the Deferrable Server and Polling Server, respectively.

It follows the same structure as the `Plot` class behind the `plot` command, and implements the new events as instance methods. In the trace the new events are passed as arguments to the `bufferplot` command, rather than `plot`, which then dispatches the appropriate `BufferPlot` method. Figure 5 shows an example of how this Grasp plugin correlates the contents of the buffers with the task execution of the application.

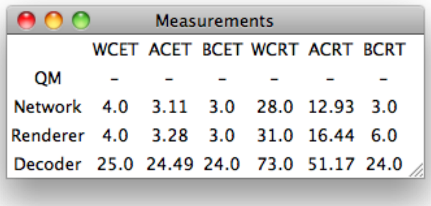
The Tcl file format of a Grasp trace makes it possible to embed plugins inside a trace file. Since a plugin is simply a definition of methods called within a trace file, inserting the plugin code at the beginning of the trace file will make sure that the necessary methods are defined before they are used. This allows to distribute a single self-contained trace file which can be visualized with any Grasp player, independent of the available plugins.

E. Grasp measurement

The Grasp player measures the execution and response time of jobs and provides a summary of the average, best case and worst case times for all jobs of a task. This information is shown on demand, by clicking on a job or a task label, or by selecting "Measurements" from the menu, shown in Figure 6.

The Grasp plugins also allow to easily implement custom measurement tools, as we did for measuring the mode change latencies for our recent work [Holenderski et al., 2009].

To measure the mode change latencies we have added a simple plugin which extended the `Plot` class with


Measurements

	WCET	ACET	BCET	WCRT	ACRT	BCRT
QM	-	-	-	-	-	-
Network	4.0	3.11	3.0	28.0	12.93	3.0
Renderer	4.0	3.28	3.0	31.0	16.44	6.0
Decoder	25.0	24.49	24.0	73.0	51.17	24.0

Fig. 6. Example of trace measurements, summarizing the worst-case (WCET), average-case (ACET) and best-case (BCET) execution times, and the worst-case (WCRT), average-case (ACRT) and best-case (BCRT) response times for all application tasks.

three new events: `latencyStart`, `latencyStop` and `latencySummary`. The first two events are generated throughout the simulation whenever a mode change occurs. The latter event is generated at the end of the simulation. Its handler collects all the latencies, uses the `gnuplot` tool [gnu, 2010] to plot them on a graph, and automatically writes the graph to a postscript file.

V. Conclusions

In this paper we presented the Grasp toolset for tracing, visualizing and measuring the behavior of real-time systems. Grasp can be used to evaluate new algorithms and scheduler implementations in an operating system. We have used Grasp extensively during the development of several extensions for the μC/OS-II real-time operating system, some of which were used in this paper to illustrate the Grasp features.

Grasp is composed of three parts: (i) the recorder, (ii) the trace file and (iii) the player. The Grasp's recorder takes the instrumentation approach to tracing, catching all events of interest. It limits the interference by storing the traced events in memory during runtime and writing the trace to a file only at the end of a run. The recorder is the only operating system specific part of the Grasp toolset. A Grasp trace is stored in the Tcl script format. The expressiveness of this format allows to easily extend Grasp functionality with visualization and measurement plugins. The Grasp player interprets the Tcl script containing the recorded trace. It visualizes task execution, task synchronization, servers in hierarchical scheduling and buffer usage for tasks communicating via shared buffers.

Future work

We have implemented a Grasp recorder for μC/OS-II within the OpenRISC simulator. Our current research focusses on (i) deploying the Grasp recorder in an embedded environment, and (ii) investigating the applicability of the Grasp recorder in other real-time operating systems.

In this paper we have shown how to visualize a Grasp trace using the Grasp player. A trace, however, can also be used for validating the behavior of the target system by comparing its trace to a reference trace automatically. To make sure that a change in the implementation of one primitive has no impact on other parts of the system, we have setup a test suite which automatically checks whether a new μC/OS-II extension did not invalidate existing behavior. The test suite is comprised of a set of test applications with reference traces and a shell script. The script executes all the test applications and compares the new traces against the reference traces. Currently, two traces are considered to be equivalent if they exhibit the same timing behavior, modulo the unique identifiers (e.g. job identifiers) particular to every simulation run. This approach, however, may result in false positives, when a trace exhibits correct behavior, but is not equivalent to the reference trace due to different overheads of the primitives resulting in different computation times of tasks and consequently leading to a different preemption behavior, which nonetheless may be correct. As future work we would like to investigate more resilient testing methods for exploiting the Grasp traces to validate the timing behavior of a real-time system.

REFERENCES

[uco, 2007] *uCOS-II WIN32, LINUX und Freescale HCS12 PORT*. 2007. URL http://www.it.fht-esslingen.de/~zimmerma/software/.

[gnu, 2010] *Gnuplot*. 2010. URL http://www.gnuplot.info.

[Baker, 1991] T. P. Baker. *Stack-based scheduling for realtime processes*. Real-Time Systems, vol. 3(1):pp. 67–99, 1991.

[Bolado et al., 2004] M. Bolado, H. Posadas, J. Castillo, P. Huerta, P. Sánchez, C. Sánchez, H. Fouren, F. Blasco. *Platform based on open-source cores for industrial applications*. In *Conference on Design, Automation and Test in Europe (DATE)*, p. 21014. 2004.

[Davis and Burns, 2005] R. I. Davis, A. Burns. *Hierarchical fixed priority pre-emptive scheduling*. In *Real-Time Systems Symposium (RTSS)*, pp. 389–398. 2005.

[Holenderski et al., 2008] M. Holenderski, R. J. Bril, J. J. Lukkien. *Using fixed priority scheduling with deferred preemption to exploit fluctuating network bandwidth*. In *Work in Progress session of the Euromicro Conference on Real-Time Systems (ECRTS)*. 2008.

[Holenderski et al., 2009] M. Holenderski, R. J. Bril, J. J. Lukkien. *Swift mode changes in memory constrained real-time systems*. In *International Conference on Embedded and Ubiquitous Computing (EUC)*, pp. 262–269. 2009.

[Labrosse, 1998] J. J. Labrosse. *Microc/OS-II*. R & D Books, 1998.

[Lehoczky et al., 1987] J. P. Lehoczky, L. Sha, J. K. Strosnider. *Enhanced aperiodic responsiveness in hard real-time environments*. In *Real-Time Systems Symposium (RTSS)*, pp. 261–270. 1987.

[McGavin et al., 2006] M. McGavin, T. Wright, S. Marshall. *Visualisations of execution traces (vet): an interactive plugin-based visualisation tool*. In *Australasian User Interface Conference (AUIC)*, pp. 153–160. 2006.

[Mughal and Javed, 2008] M. I. Mughal, R. Javed. *Recording of Scheduling and Communication events on Telecom Systems*. Master's thesis, Mälardalen University, 2008.

[OpenCores, 2009] OpenCores. *OpenRISC overview*. 2009. URL http://www.opencores.org/project,or1k.

[Strosnider et al., 1995] J. K. Strosnider, J. P. Lehoczky, L. Sha. *The deferrable server algorithm for enhanced aperiodic responsiveness in hard real-time environments*. IEEE Transactions on Computers, vol. 44(1):pp. 73–91, 1995.

[Welch et al., 2003] B. Welch, K. Jones, J. Hobbs. *Practical Programming in Tcl and Tk*. Prentice Hall, 2003.

[Whitham and Audsley, 2006] J. Whitham, N. Audsley. *MCGREP–a predictable architecture for embedded real-time systems*. In *Real-Time Systems Symposium (RTSS)*, pp. 13 –24. 2006.