Journal of Universal Computer Science, vol. 15, no. 8 (2009), 1563-1588
submitted: 4/11/08, accepted: 23/4/09, appeared: 28/4/09 © *J.UCS*

Tools and Stochastic Metrics for Debugging Temporal Behaviour of Real-Time Systems

Joaquín Entrialgo
(University of Oviedo, Spain
joaquin@uniovi.es)

Javier García
(University of Oviedo, Spain
javier@uniovi.es)

José Luis Díaz
(University of Oviedo, Spain
jldiaz@uniovi.es)

Daniel F. García
(University of Oviedo, Spain
dfgarcia@uniovi.es)

Abstract: In real-time systems, temporal behaviour is as important as functional behaviour, so several techniques have been especially developed for these systems. Stochastic analysis techniques model the execution time of tasks as random variables and constitute a very powerful tool to study the temporal behaviour of real-time systems. However, as they can not avoid all the timing bugs in the implementation, they must be combined with measurement techniques in order to gain more confidence in the implemented system. This paper presents a monitoring tool which can measure real-time systems developed using POSIX. The corresponding analysis and a visualization tool that makes it possible to find errors easily is also introduced. In order to find bugs in the timing behaviour of the system when an stochastic analysis technique is used, two metrics, called "pessimism" and "optimism", are proposed. They are based on two random variables, the optimistic and the pessimistic execution time, which are also introduced in this paper. These metrics are used in the debugging tools to compare the model and the measured system in order to find errors. The metrics are examined in four case studies.

Key Words: Debugging aids, Real-time Systems, Monitors
Category: C.3, D.2.4, D.2.5

1 Introduction

What makes real-time systems different from other types of systems is that the temporal behaviour is as important as the functionality. This begins with the clear definition of timing constraints, usually in the form of deadlines and periods for the different tasks that the system must carry out. Thus, special techniques are needed to guarantee the timing constraints and to debug the temporal behaviour of real-time systems.

For the first goal, guaranteeing the timing constraints, several analytical methods have been proposed. Traditional techniques, such as the processor utilization analysis [Liu and Layland 1973, Lehoczky 1990] and response time analysis [Tindell et al. 1994], use a model of the system where the execution time of the tasks is represented by their worst case execution time (WCET). Using this value, these analyses can obtain an upper bound for the response time of the tasks. Thus, it is possible to determine if all of the tasks will fulfill their deadlines even in the worst circumstances. However, in modern systems, the great variability of the execution times, due to caches, out of order execution, etc., results in excessively pessimistic WCETs, that is, the WCET is much greater than the average case and its frequency of occurrence is negligible, which leads to oversized systems.

To overcome this problem, another set of techniques for guaranteeing the timing behaviour of real-time systems have been developed [Abeni and Buttazzo 2001, Manolache et al. 2001, López et al. 2008]. These techniques, called probabilistic techniques, analyse the timing properties of the system based on a stochastic model of its timing properties. Rather than using only the WCET, these techniques model execution time with a probability function which assigns probabilities to each possible execution time. From these execution times, the analysis computes the response time for each task, which is also a probability function. The response time is computed taking into account the fact that the tasks share a CPU. The probability of fulfilling the deadlines can be obtained from the probability distributions of the response time.

In addition to analytical methods, measuring and debugging the temporal behaviour of the system is needed, because any mistake in the model or in the application of the techniques can lead to an error in the system.

One important step in removing the errors from a system, i.e., debugging it, is finding where the errors lie. In the context of timing analysis, there is a model that guarantees the absence of errors, so if there is an error in the system, there must be an error in the model. Thus, identifying where the error lies means finding which parameter or parameters of the model do not reflect reality. In order to find errors in probabilistic parameters, probability functions must be compared. In this paper the problems involved in this process are explored and two metrics are proposed to solve them.

The rest of the paper is organized as follows: Section 2 presents related work; Section 3 gives a general vision of the debugging strategy, including the system model and the toolset where the metrics are used; in Section 4 the problems of debugging a system analyzed with stochastic models are introduced and the metrics to solve them are defined; Section 5 examines the metrics through case studies; and finally, Section 6 summarizes the most important contributions of this paper.

2 Related Work

Much work has been done dealing specifically with monitoring and debugging real-time systems. A compilation of the most significant work up to 1995 can be found in [Tsai and Yang 1995] and [Tsai et al. 1996].

[LeBlanc and Mellor-Crummey 1987] proposed a technique called *Instant Replay*, that was used later in [Dodd and Ravishankar 1992] and [Thane et al. 2003]. This technique is used to debug functional errors, not temporal errors like the ones addressed in this paper. [Tokuda et al. 1988] is focused in detecting deadline misses, but it does not relate them with other parameters of the model. [Chodrow et al. 1991] analyses how to check the temporal specifications in runtime, but it uses a time-driven approach in the gathering of events that, as [Petters 2002] and [Stewart and Arora 2003] point out, is not well-suited to real-time systems. The monitoring system presented in [Raju et al. 1992] has a similar goal to the one in [Chodrow et al. 1991], but using an event-driven approach. Its main drawback is that it uses a specification based on RTL (Real-Time Logic), a technique scarcely used in practice. This is the same problem with JRMT [Mok and Liu 1997], which extends their previous work.

[Timmerman et al. 1993] introduces the problem of detecting temporal errors in real-time systems when a trace can have a huge amount of data. In order to overcome this problem, an expert system is proposed, but it requires expressing the requirements in Prolog, which reduces its usability.

[Wilner 1995] describes WindView, a commercial tool developed by Wind River for their operating system VxWorks. It is a tool that gathers a great amount of data, but does not provide an automatic analysis to find errors. Furthermore, it does not relate its measurements with the model used for schedulability analysis, a problem also present in JewelNT [Gergeleit and Nett 1999], a monitoring system for real-time systems in Windows NT; and in [Yaghmour 2000], which explains an instrumentation system in a Linux Kernel extension for real-time systems, RTAI (Real-Time Application Interface). [Petters 2002] measures blocks of code in order to feed stochastic models, but is not aimed at debugging a working system. [Terrasa and Bernat 2003] shows how to obtain metrics from a trace, but does not take stochastic techniques into account. [Stewart and Arora 2003] aims to find bugs comparing measurements and models, but uses a non-stochastic model, with a very restricted implementation.

None of the previous work addresses the main contribution of [Entrialgo et al. 2007]: debugging systems analyzed with stochastic metrics. Furthermore, none of the above papers propose a portable implementation in POSIX. This paper extends [Entrialgo et al. 2007] with an improved introduction, more analysis of related work, an extended description of the monitoring tool, including data about intrusiveness, and a new case study.

3 Debugging Strategy

3.1 Overview

In order to understand how the metrics presented in this paper work, it is necessary to know the general debugging strategy in which they are included. The strategy addresses timing —not functional— errors. These are defined as follows:

Definition 1. A **timing error** is a non-fulfillment of the timing requirements of the system, or an optimistic deviation from the model which guarantees the requirements.

Notice that this definition emphasizes two different kind of errors. On the one hand, there are errors that are non-fulfillments of the specifications. On the other hand, there are errors that are differences between the timing behaviour of the system and of the model in a way that makes the model optimistic, that is, makes the model obtain higher probabilities of fulfilling the deadline than the real probability, so there is a false security of fulfilling the specifications.

As shown in Figure 1, the proposed debugging strategy works by measuring an implementation of the system and comparing the values obtained with the values in the model. In order to make this comparison, three elements must be available: a model of the system, an implementation, and a monitoring tool. In addition, to make debugging easier, the measurements obtained from the monitor should be analyzed and presented with a graphical tool. The final goal of the strategy to find errors and pinpoint in which tasks and in which parameters of the tasks they occur. The metrics presented in this paper, which are calculated by the analysis and visualization tool, help in making the comparison between the measurements and the model in order to find the errors.

Although the basic principles of the strategy are applicable to any real-time system, in order to test it and demonstrate its capabilities a definition of the model, the implementation, the monitor, and the analysis and visualization tool must be chosen. In the following sections these elements are described.

3.2 System model

Stochastic analysis techniques are used as a base for analysing the measurements, so one of them must be chosen. The one presented in [Kim et al. 2005] has been selected because it provides the most exact analysis without needing strong restrictions, as shown in [Díaz et al. 2002].

The system is composed of a set of n independent periodic tasks $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_n\}$. Without loss of generality, we assume that the tasks are sorted in decreasing order of priority. Each task, τ_i, is defined by the tuple $(T_i, \Phi_i, J_i, \mathcal{C}_i, D_i, M_i)$, where T_i is the period of the task, Φ_i is its initial phase,

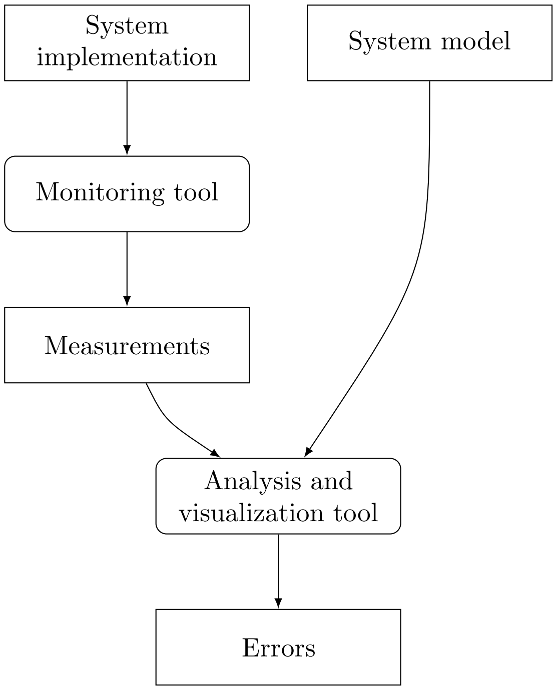


Figure 1: Overview of the debugging strategy

J_i is its release jitter, $\mathcal{C}_i$ is its execution time, D_i is its deadline and M_i is its maximum allowable ratio of deadline misses.

The execution time, $\mathcal{C}_i$, is a discrete random variable which assigns probabilities to the possible execution times of the task. It can be defined with a probability function denoted by $f_{\mathcal{C}_i}(\cdot)$, where $f_{\mathcal{C}_i}(c) = \mathbb{P}\{\mathcal{C}_i = c\}$, i.e., $f_{\mathcal{C}_i}(c)$ is the probability of the execution time being c. Alternatively, the execution time distribution can also be defined using its cumulative distribution function (CDF), denoted by $F_{\mathcal{C}_i}$, where

$$F_{\mathcal{C}_i}(c) \triangleq \mathbb{P}\{\mathcal{C}_i \leq c\} = \sum_{i=-\infty}^{c} f_{\mathcal{C}_i}(i) \tag{1}$$

$F_{\mathcal{C}_i}$ is a monotone increasing and right-continuous function. Furthermore, $\lim_{c\to\infty} F_{\mathcal{C}_i}(c) = 1$.

For example, consider the model of task τ_0 presented in Table 1. Figure 2 shows a Gantt diagram representing four activations of the task. The arrows indicate the arrivals of the task. The time between the beginning of the system execution and the first arrival of the task is its phase, Φ_0. The time between two arrivals is the period of the task, T_0. Triangles are used to represent the end of the deadline of each task arrival; thus, the interval between a task activation and a triangle is its deadline, D_0. Grey rectangles represent the execution of the task. The interval between an arrival and the moment the task begins to execute is the release jitter for that activation; the maximum of all these jitters is J_0. Finally, the interval from the start of a task execution to its end is the computation time of that activation, c.

Task	T_i	D_i	Φ_i	J_i	M_i	c	$f_{C_i}(c)$
τ_0	40	35	20	10	0.1	10	0.8
						15	0.2

Table 1: Task model example. All times are in ms

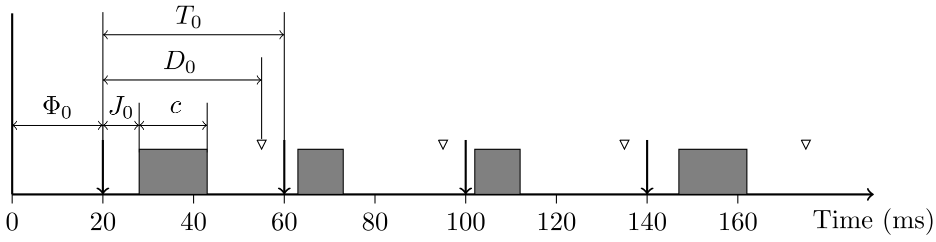


Figure 2: Gantt diagram of an example task

Using the equations in [Díaz et al. 2002], the probability function of the response time of each task can be computed. From this probability function, the probability of missing the deadline according to the model can be computed. If this probability is greater than the corresponding maximum allowable ratio of deadline misses for all tasks, the system is feasible according to the model, i.e., the model guarantees that the system will fulfill its requirements. A summary of how the analysis works, along with a step-by-step example of a calculation of a job random response time can be found in [Kim et al. 2005].

3.3 System Implementation

An implementation based on POSIX [IEEE 2004] has been chosen because it is the most common standard in real-time operating systems, so it provides a wide applicability of the ideas presented in this paper.

In order to identify all the elements of the model in the measured system, some constraints must be applied to the implementation. Firstly, in POSIX the most common ways to implement a real-time system that follows the model presented in the previous section are several processes with one thread or one process with several threads. We have chosen the second alternative because POSIX has protocols to avoid priority inversion problems that arise when the model is extended to share resources between tasks.

In the proposed implementation there is a master thread that creates a thread for each task in the model and then goes to sleep. Each of the created threads mounts a timer with its period that generates a signal when it expires. Each thread waits for its signal and, when it is received, the thread carries out the computation corresponding to one of its jobs.

3.4 Monitoring tool

In order to measure the system implementation presented in the previous section, a monitoring tool has been developed. Portability across POSIX real-time operating systems was one of its main objectives. There is a tracing standard in POSIX, but it does not standardize what to trace; it simply standardizes the interface to event handling functions. For example, it provides a function for opening a log, but it does not determine which events must be captured in every POSIX compliant operating system.

Furthermore, as the section dedicated to tracing in POSIX is optional and it has not been widely implemented by real-time operating systems, it was not selected for use in this work. The proposed monitoring tool works at source-code level, as this is standardized by POSIX. In this way, the monitoring tool can be used for any application written for any POSIX compliant operating system.

The monitoring tool instruments the source code by adding instructions which capture the occurrence of the events needed to debug the system. The events always have an associated timestamp and may have some other parameters depending on their type. The instrumentation is carried out by means of C macros which must be included in the source code of the application.

An important point in order to give meaningful results to the analyst is relating the information gathered with the analysis model. Typically, in analysis the tasks are assigned a name, for example, *Check_Sensor*. This information is lost when the system is executed, as POSIX uses its own numerical identifiers for threads. These identifiers can be different in each execution, so task *Check_Sensor* can be thread 4 in one execution and thread 12 in another. In order to relate the events in each thread with the data of a task in the model, an instrumentation function indicating the task associated with each thread has to be inserted.

To avoid inserting new context switches in the system, which would increase the monitoring overhead, the monitoring tool does not introduce more threads in the application. Its instructions are executed in the thread where events are gathered. In addition, in order to avoid a shared resource between all the threads, which would significantly change the system model, there is an independent event buffer for each thread.

At the end of the system execution, the events from all the threads are saved to disk, thus becoming traces that can be analyzed and visualized by the tool described in the next section.

The timestamps associated with each event are gathered with the POSIX function *clock_gettime()*. The maximum error in a timestamp depends on the precision of the clock available with this function, which is implementation dependent. In the test platform described in Table 3, it is 0.499504 milliseconds.

The intrusiveness of the monitoring tool is implementation dependent, but it also depends on the characteristics of the measured system: the higher the event rate, the higher the intrusiveness. As task activations are the most important event that must be gathered in order to measure computation times and the start and end of tasks, the event rate is directly related to the frequency of task activations, which is the inverse of their period. Therefore, for systems where the tasks have small periods, the intrusiveness is bigger. In the test platform described in Table 3, the maximum time taken by the monitoring system to gather an event is 0.03878 milliseconds.

The monitoring system also introduces overhead in the form of memory consumption, and it is also implementation dependent. It has a constant value that arises from the code of the instrumentation functions. In the test platform, it was found to be 9.85 KBytes. Additionally, there is a variable factor which depends on the number of tasks and the number of measured events. In the test platform, 28 bytes are needed for the basic information of each task and each measured event requires 20 bytes.

3.5 Analysis and visualization tool

The analysis and visualization tool reads both the traces generated by the monitoring tool and the model used in the schedulability analysis. With this information, the analysis and visualization tool carries out an analysis looking for errors and then presents information to the user in order to help debug them.

In order to address the problem of showing the analyst the great amount of data contained in the trace, a hierarchical approach is used. First, the tool shows a summary of the system state in the “Metrics Windows” (an example for a case study can be seen later in Figure 6), which, for each task, provides a series of metrics, based on the system model, the measurements and a comparison of both. One of the main areas of the window shows a list of the problems found. The user can further investigate these problems with the help of the metrics and by using supplementary windows with other graphs, such as Gantt diagrams or probability functions. Gantt diagrams are extended to present the main events related to the computing model, as shown in [Entrialgo et al. 2003].

In order to find errors, the tool compares the values in the model, which include the specifications, and values obtained from the trace. This comparison is easy to carry out when the analysis is used with deterministic techniques. For instance, in order to find an error in the WCET just two values, the WCET in the model and the WCET in the measurements, must be compared. When stochastic models are used in the analysis, the comparison becomes more difficult, as probability functions must be compared. The following section addresses this issue.

4 Comparison of stochastic values

When a stochastic model is used to analyze a real-time system, the execution time of each task is characterized as a random variable. As proved in [Díaz et al. 2004], the analysis guarantees the deadline miss ratio as long as this random variable follows a distribution which is more pessimistic than the real distribution of the execution time. In order to compare the pessimism in two distributions, the "worse than" relationship must be introduced:

Definition 2. Given two random variables, $\mathcal{X}$ and $\mathcal{Y}$, we state that "$\mathcal{X}$ is worse than $\mathcal{Y}$", and denote it as $\mathcal{X} \succcurlyeq \mathcal{Y}$ if $F_{\mathcal{X}}(x) \leq F_{\mathcal{Y}}(x)$ for all x.

If $\mathcal{X}$ and $\mathcal{Y}$ are two distributions of execution time and $\mathcal{X} \succcurlyeq \mathcal{Y}$, this means, intuitively, that $\mathcal{X}$ assigns lower probabilities to lower times than $\mathcal{Y}$, thus $\mathcal{X}$ is more pessimistic as it assigns greater probabilities for longer times, meaning it will take longer to complete the task. Graphically, $\mathcal{X} \succcurlyeq \mathcal{Y}$ means that the curve of $F_{\mathcal{X}}(\cdot)$ is always below the curve of $F_{\mathcal{Y}}(\cdot)$.

In a real-time system, there will be a timing error as defined in Definition 1 when the model distribution is not more pessimistic than the real one. It must be noted that when a system is measured for a limited amount of time, the monitor obtains a series of values for the execution time of each task. A distribution function can be built from these values, but rather than the real distribution of the execution time, it will only be a sample. What the analysis and visualization tool must do is to infer from this sample distribution whether the model distribution is more pessimistic than the (unknown) real distribution.

Task	$T_i = D_i$	Φ_i	J_i	M_i	c	$f_{C_i}(c)$
τ_0	200	0	0	0.1	20	0.8
					100	0.2

Table 2: Example model. All times are in ms

In order to show this problem, an example will be introduced. Let us consider the model presented in Table 2.

Figure 3 shows the plot of three example CDFs (Cumulative Distribution Functions) for task τ_0: the model distribution, the measured distribution obtained from a measurement session, and the real distribution. As can be seen, the model is more pessimistic than the real distribution, as the model CDF is always below the real CDF. Therefore, there is no error. On the other hand, the measurements CDF is not equal to the real CDF, as it is a sample of finite size. The decision as to whether there is error is made by comparing the CDF of the measurements and the model CDF, because the real CDF is unknown. In this example, the model is not more pessimistic than the CDF of the measurements — there are parts of the model curve that go above the measurement curve.

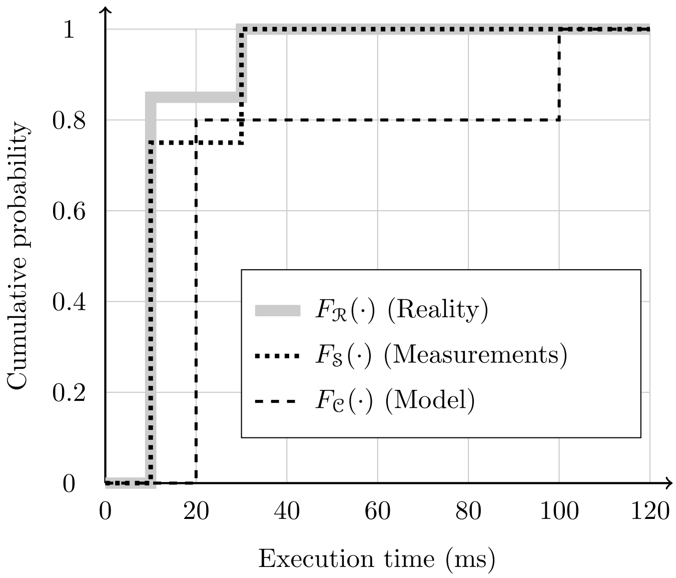


Figure 3: Probability distribution example

What this example shows is that the decision as to whether there is an error can not be made by simply assessing whether the model CDF is always below the CDF of the measurements : the sampling error can make the CDF of the measurements go below the model CDF even when the real CDF is always above the model CDF. In spite of this, with a high enough number of measurements, the CDF of the measurements and the real CDF should be very close; thus, in a system without timing errors there should be few intervals where the CDF of the measurements is below the model CDF.

The classic solution to this problem —which appears in other fields such as economics— is to use the Kolmogorov-Smirnov test. Unfortunately, this test requires the distributions to be continuous and the sample, independent. In our case the distributions are discrete and, due to caches and other architectural components, the sample is not independent.

Instead of a statistical test, in this work we propose using heuristic metrics. The goals pursued in defining these metrics are as follows:

- The metrics should have threshold values that indicate when an error is found.
- When there is an error, the metrics should help in determining in which task the error lies. In order to accomplish this goal, the metrics for different tasks should be comparable.
- The metrics should have a graphical interpretation that is easily understandable for the analyst.
- The metrics should have a simple formulation.

After testing several metrics with different case studies (see Section 5), two complementary metrics, pessimism and optimism, have been chosen. They are based on comparing the mean value of the CDF of the measurements and the model CDF, but this comparison must take into account the fact that optimism for some execution times can not be compensated for with pessimism for other execution times. In order to avoid this compensation, two new random variables must be introduced.

Let $\mathcal{C}$ be the random variable which characterizes the execution time in the model for a task and let $\mathcal{S}$ be the random variable which characterizes the execution time according to the measurements.

Definition 3. The **optimistic execution time** $\mathcal{C}^{\mathbf{O}}$ is defined as the random variable which has the following CDF:

$$F_{\mathcal{C}^{\mathrm{O}}}(x) = \max\{F_{\mathcal{C}}(x), F_{\mathcal{S}}(x)\} \tag{2}$$

Definition 4. The **pessimistic execution time** $\mathcal{C}^{\mathbf{P}}$ is defined as the random variable which has the following CDF:

$$F_{\mathcal{C}^{\mathrm{P}}}(x) = \min\{F_{\mathcal{C}}(x), F_{\mathcal{S}}(x)\} \tag{3}$$

Figures 4 and 5 show a graphical interpretation of the CDF of these variables by means of an example. As seen in Figure 4, the CDF of the optimistic execution time is the curve that always follows the highest of the CDF of the measurements and the model CDF. Similarly, as seen in Figure 5, the CDF of the pessimistic execution time is the curve that follows the lowest of the CDF of the measurements and the model CDF.

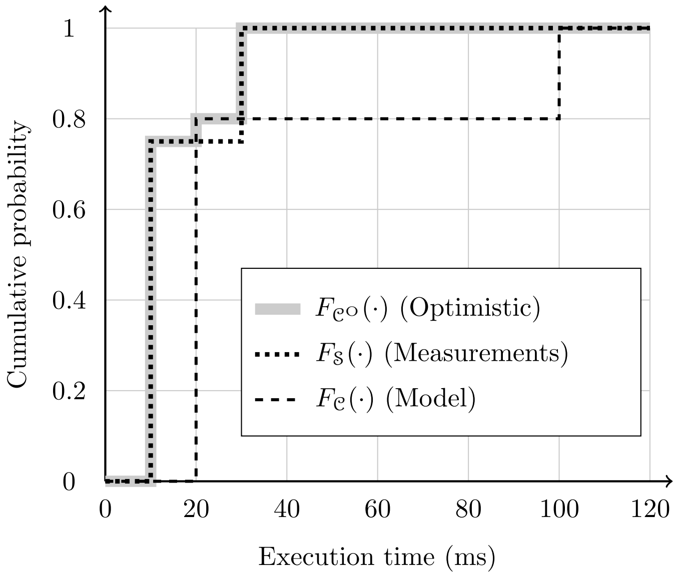


Figure 4: Optimistic execution time example

To obtain a metric of the optimism in the model, the differences between the mean of the optimistic execution time and the measured execution time are used. By using the optimistic execution time, there is no compensation of optimism with pessimism. In a similar way, pessimism is defined as the difference between the mean of the pessimistic execution time and the measured execution time.

Next, the metrics are formally defined.

Definition 5. **Optimism** is defined as:

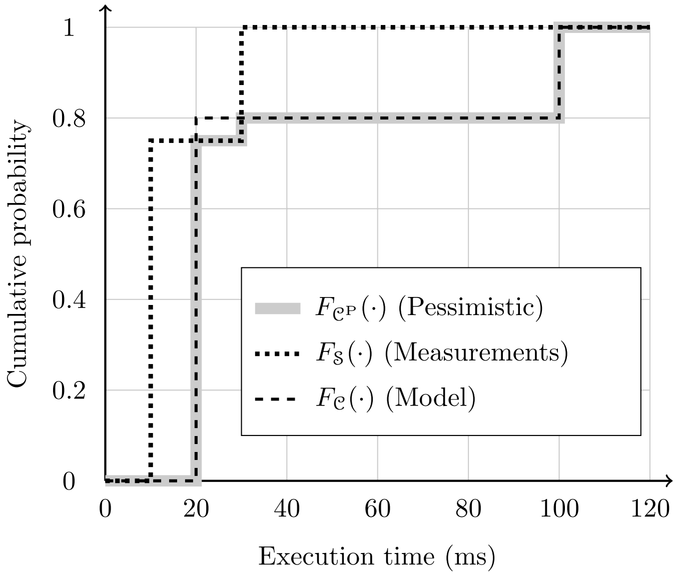


Figure 5: Pessimistic execution time example

$$O = \frac{\bar{S} - \bar{C}^{\mathrm{O}}}{\bar{S}} \tag{4}$$

Definition 6. Pessimism is defined as:

$$P = \frac{\bar{C}^{\mathrm{P}} - \bar{S}}{\bar{S}} \tag{5}$$

In the previous equations, $\bar{S}$, $\bar{C}^{\mathrm{O}}$ y $\bar{C}^{\mathrm{P}}$ are the mean of the random variables $\mathcal{S}$, $\mathcal{C}^{\mathrm{O}}$ and $\mathcal{C}^{\mathrm{P}}$ respectively.

As seen in Equations 4 and 5, the difference between means is divided by the mean of the measured execution time. This is done in order to obtain a relative metric which can be easily understood by the analyst and comparable between different tasks. Multiplying by 100 to convert the values to percentages, it can be said, for instance, that in a task there is a pessimism of 15% and an optimism of 3%.

When pessimism is positive and optimism is zero, the model CDF is always below the CDF of the measurements. This is good, as it indicates that the model is based on pessimistic estimations of the execution times. Conversely, finding a low pessimism and a high optimism is bad, as it indicates the model is based on optimistic estimations of the execution time and, therefore, the analysis can not guarantee the temporal specifications.

5 Case studies

In order to show the utility of the metrics and to test if they fulfill the goals previously stated, this section provides four case studies. All of the case studies have been implemented in a platform with the characteristics shown in Table 3. The platform uses QNX as its operating system, which is one of the main real-time operating systems and follows the POSIX standard. In order to implement a controlled set of tasks for the case studies, synthetic tasks have been used. Computation times are emulated with loops, and probabilities with conditional statements using random numbers.

Hardware	
Processor	Pentium III
Processor Frequency	800 MHz
Front Side Bus Frequency	133 MHz
Cache L1	16KB/16KB
Cache L2	256KB
Main memory	256MB
Software	
Operating System	QNX 6.2.1
Compiler	gcc 2.95.3

Table 3: Test platform

5.1 Case study 1: Determining which task contains the error

This case study shows how the metrics work within the complete debugging strategy and how they help in finding in which task the error lies. A system made up of four tasks has been implemented in the test platform detailed in Table 3. Table 4 shows the model of this system. In the model, the execution times have been increased by 10% in order to emulate the pessimism of the models. The 0.499504 ms of jitter is due to the precision of the operating system clock. Although the phase, Φ_i, is useful in order to have different reference points for the periods of each task, which might control processes with different offsets, in all the case studies the phase will be zero, as it does not affect the metrics.

As seen in Table 5, according to the analysis of the model with the techniques presented in [Díaz et al. 2002], all of the tasks will fulfill their maximum allowable ratio of missed deadlines.

Task	$T_i = D_i$	Φ_i	J_i	M_i	c	$f_{C_i}(c)$
τ_0	100	0	0.499504	0.1	11	1
τ_1	200	0	0.499504	0.1	22	0.8
					110	0.2
τ_2	300	0	0.499504	0.1	33	0.1
					55	0.9
τ_3	400	0	0.499504	0.1	11	0.1
					33	0.5
					121	0.39
					121.1	0.01

Table 4: Model for the case study 1. All times are in ms

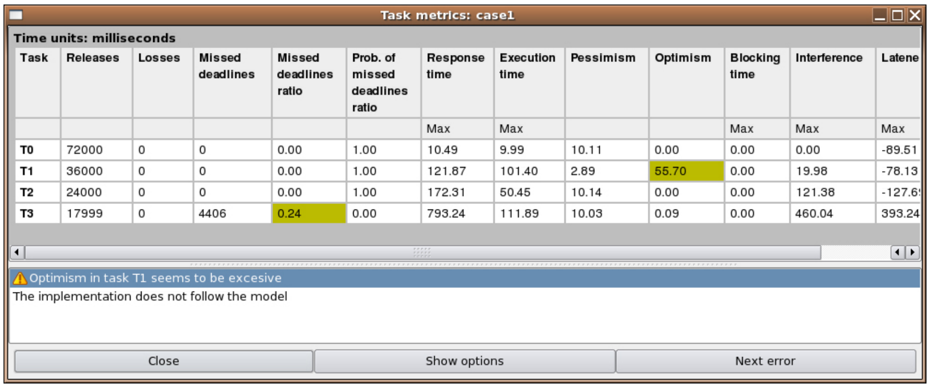
Task	M_i	**Probability of missing the deadline**
τ_0	0.1	0
τ_1	0.1	0
τ_2	0.1	0
τ_3	0.1	0.05697

Table 5: Model analysis results for the case study 1

An error in the implementation of the system was deliberately introduced. The two possible execution times of task τ_1 come from a conditional statement and its condition was reversed. This case study assesses whether the debugging tools are able to find the error.

The implementation was measured for two hours using the monitor described in Section 3.4. The measurement interval was selected after assessing that the metrics converge to a steady value, using the window from the analysis and visualization tool with the evolution of pessimism and optimism presented later, in Section 5.2. The size of the resulting trace was 8.54MB.

After opening the corresponding trace with the analysis and visualization tool described in Section 3.5, the "Metrics Window" shown in Figure 6 presents a summary of the system behaviour.


Task metrics: case1

Time units: milliseconds

Task	Releases	Losses	Missed deadlines	Missed deadlines ratio	Prob. of missed deadlines ratio	Response time	Execution time	Pessimism	Optimism	Blocking time	Interference	Latene
						Max	Max			Max	Max	Max
T0	72000	0	0	0.00	1.00	10.49	9.99	10.11	0.00	0.00	0.00	-89.51
T1	36000	0	0	0.00	1.00	121.87	101.40	2.89	55.70	0.00	19.98	-78.13
T2	24000	0	0	0.00	1.00	172.31	50.45	10.14	0.00	0.00	121.38	-127.6
T3	17999	0	4406	0.24	0.00	793.24	111.89	10.03	0.09	0.00	460.04	393.24

Optimism in task T1 seems to be excesive

The implementation does not follow the model

Close | Show options | Next error

Figure 6: Metrics of Case Study 1

The Metrics Window is divided in to two main panels. The top panel contains a table with metrics of each task. The bottom panel presents warning and error messages obtained from an automatic analysis of the traces. One of the problems of performance debugging is finding the relevant data from among a huge amount of information. An automatic analysis searching for bugs is carried out and, as a result, the cells with data that pinpoint errors are highlighted in the table, and messages guiding the analyst to further inspection of the problem are generated.

In this case study, the analysis and visualization tool highlights the cell with the missed deadline ratio for task τ_3 (called "T3" in the tool) and the optimism in task τ_1. The missed deadline ratio for task τ_3 is more than double its specified maximum allowable ratio of deadline misses (0.1, as shown in Table 4).

The analysis predicted that, according to the model, all the tasks would fulfill their specifications, so the cause of the problem must be related to differences between the model and the measured system. The challenge is to find where. The metrics help in this regard by pointing not to task τ_3, which is where the problem appears, but to task τ_1. As can be seen, the pessimism in all the tasks except in τ_1 is close to 10% which was to be expected. Furthermore, the optimism in all the tasks is zero or very close to zero except in task $\tau 1$, where it is 55.70%.

This very high value can be further inspected with another window of the analysis and visualization tool that presents the CDFs of the model and the measurements. Figure 7 shows this window for task τ_1.

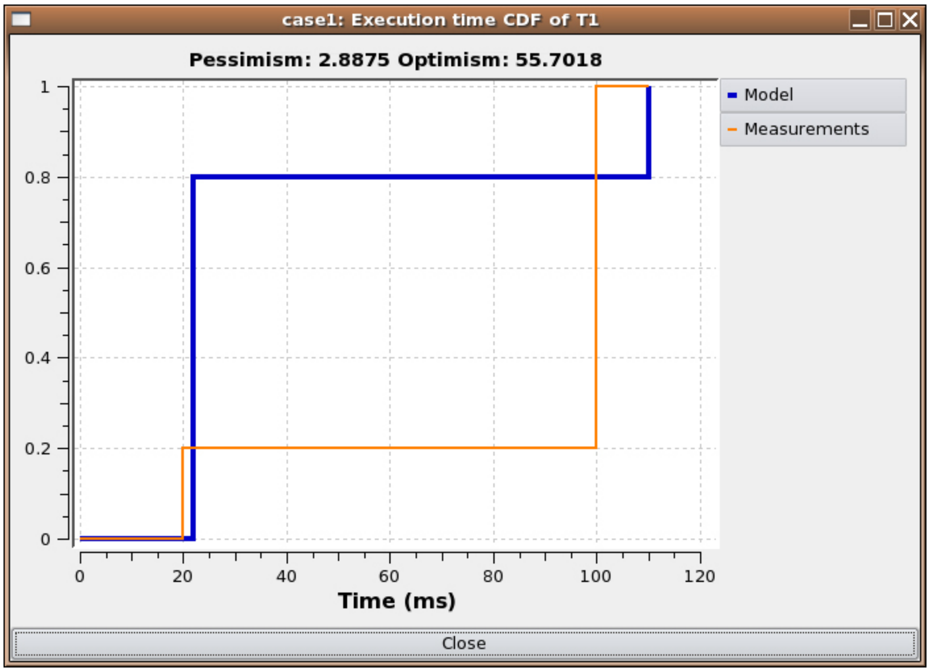


Figure 7: CDFs of task τ_1 of Case Study 1

In this figure the error that was introduced is evident: the two probabilities in task τ_1 are interchanged. Thus, the tool has helped in finding an error otherwise very difficult to locate.

5.2 Case study 2: Analysis of a system without errors

In the previous case study the model was designed to have 10% pessimism. However, as seen in Figure 6, the pessimism in the tasks that have no errors is not exactly 10%. In fact there is a small degree of optimism in task τ_3. In order to understand why this happens, and also to show the influence of sample size in the metrics, another case study has been developed. In this case study, the model aims to emulate the implementation, so that neither pessimism nor optimism is introduced. The model is shown in Table 6.

Task	$T_i = D_i$	Φ_i	J_i	M_i	c	$f_{C_i}(c)$
τ_0	100	0	0.499504	0.1	10	1
τ_1	200	0	0.499504	0.1	20	0.8
					100	0.2
τ_2	300	0	0.499504	0.1	30	0.1
					50	0.9
τ_3	400	0	0.499504	0.1	10	0.1
					30	0.5
					100	0.39
					300	0.01

Table 6: Model for the case study 2. All times are in ms

A system with these parameters was measured for two hours. Figure 8 shows a window from the analysis and visualization tool with the evolution of pessimism and optimism for task τ_0 computed when the number of measured releases of the task increases. Two issues arise from this figure:

- The values do not change as the sample size grows. The reason for this is that, both in the model and in the measurements, the task has only one execution time.
- Optimism is zero, as expected; however, pessimism is not zero but close to 0.1%. This is caused by the measurement error. The execution time in the model is 10 ms. However, the clock resolution of the test platform is 0.499504 ms, so that the closest measurable values to 10 ms are 9.99008 and 10.489584 ms (resulting from measurements of 20 and 21 clock ticks, respectively). In this case the value obtained is the former, so the measured execution time is slightly under 10 ms. Consequently, some pessimism appears.

Figure 9 shows the evolution of the metrics for task τ_1, which is very different from that of task τ_0. Firstly, there is a variation of the metrics as the sample size grows. When few releases have been measured, the values of the metrics obtained are very different from the true value, zero. However, after measuring more than a thousand releases, the metrics have values very close to zero. After the 2000$^{\text{th}}$ release, pessimism is constantly zero and optimism varies slightly at

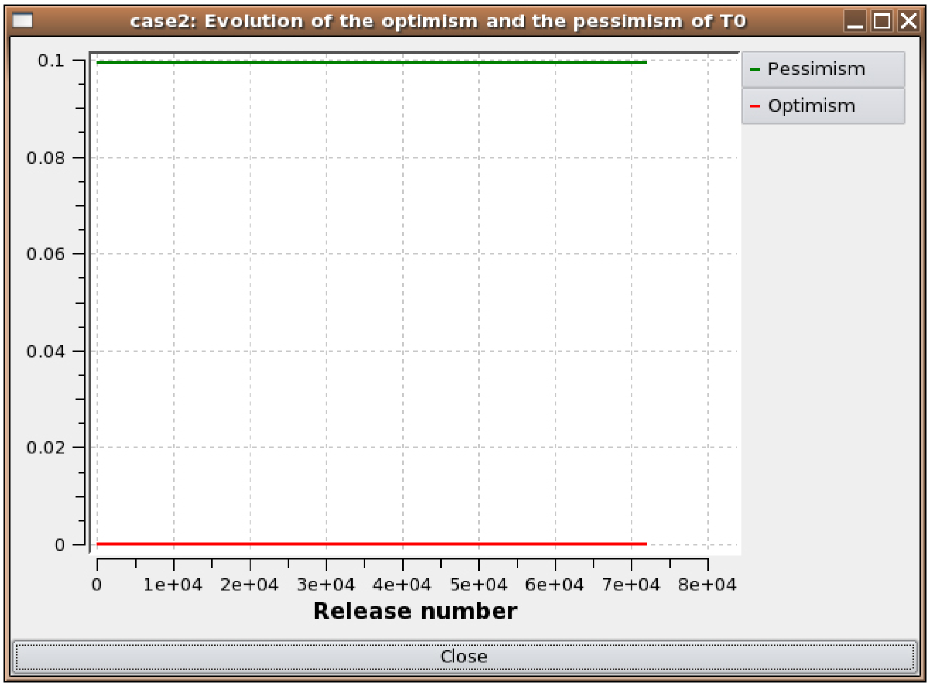


Figure 8: Evolution of the metrics for τ_0 of Case Study 2

a value close to 0.1%. As shown in the analysis for task τ_0, this is due to the measurement error.

The evolution of the metrics for tasks τ_2 and τ_3 (which is not included in this paper for the sake of brevity) is similar to that of task τ_1. Thus, this case study has shown that the measurement error has an influence on the value of the metrics, and that it is necessary to study the window with the evolution of the metrics in the analysis and the visualization in order to assess whether the metrics have reached a stable value.

5.3 Case study 3: Analysis of distributions with the same mean

As the metrics are based on the mean of the distribution, it is important to observe what happens when the model and the real distribution have the same mean, but different shapes. It must be noted that in this case there is an error, as the model and the real distribution curves will cross. Therefore, the model distribution can not always be below the real distribution, i.e., the model can not be pessimistic.

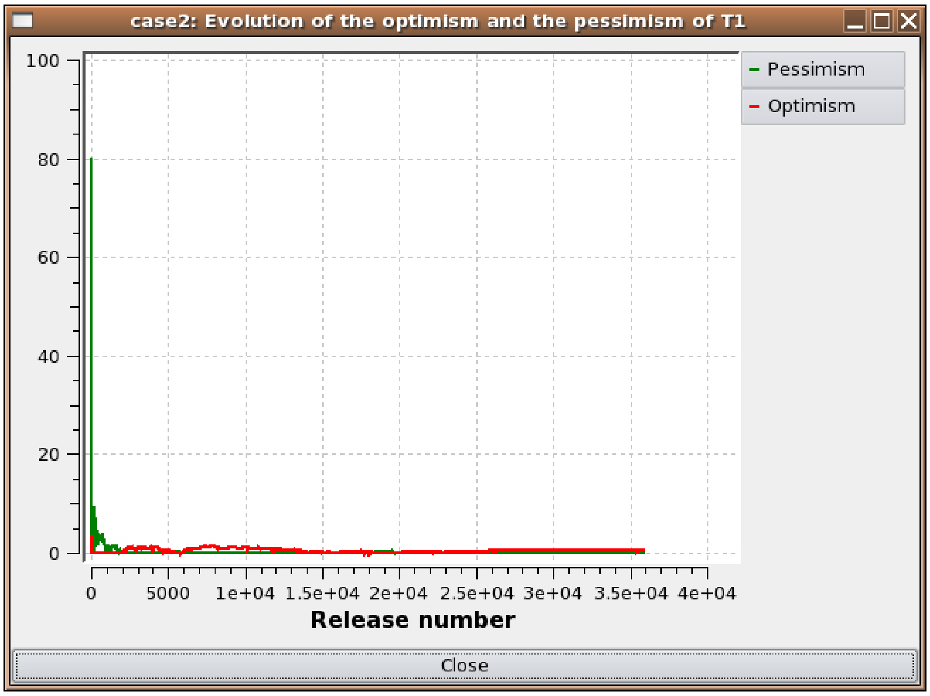


Figure 9: Evolution of the metrics for τ_1 of Case Study 2

In order to test the metrics in these circumstances, a case study with only one task has been used. The execution time of the task has been chosen so that it follows a beta distribution, as this can be easily modified to have the same mean, but different shape. Furthermore, with the right parameters, it is well suited to model execution times. The beta distribution is originally a continuous distribution between 0 and 1. In order to make the experiments, a beta distribution with parameters $\alpha = 2.5$ and $\beta = 5$ has been discretized in 20 values and scaled so that its range is $[200, 300]$ ms.

The system was measured for two hours. In order to test the metrics, ten models have been generated, each one with a different beta distribution. These beta distributions were obtained by varying the β parameter between 1 and 10 in steps of 1, and computing the corresponding α parameter so that the mean of the distribution did not change. Figure 10 shows the probability function of the original beta function ($\alpha = 2$, $beta = 5$) and of two additional models. As can be seen, the shapes of the function vary; however, their means are the same.

In order to test the metrics, the ten different models have been analyzed

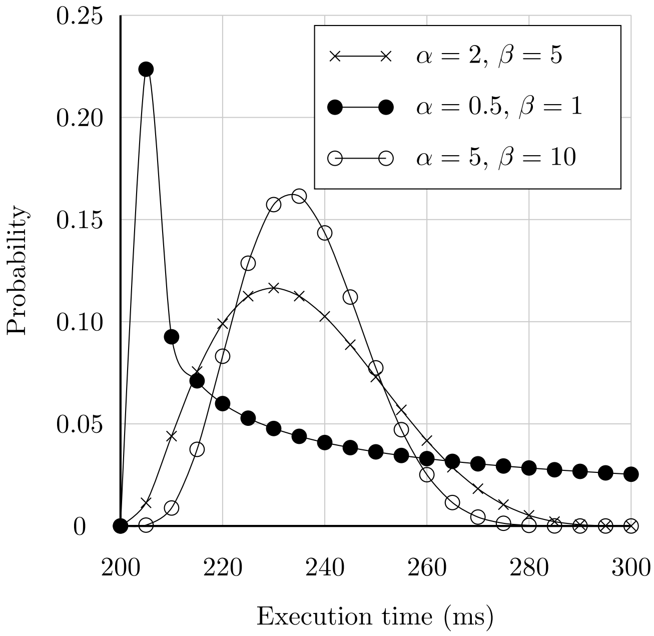


Figure 10: Beta functions used in case study 3

with the analysis and visualization tool using the same trace obtained with the original beta function. Figure 11 shows the values of the metrics as the difference between the variation coefficient of the model and the measurements changes. The variation coefficient has been chosen because it expresses the relationship between the standard deviation and the mean of the distribution. This figure shows that, even with the same mean, a small variation with respect to the mean has a significant impact on the metrics. For a difference of 0 between the variation coefficient of the model and the measurements, both metrics have a value very close to 0. When the variation coefficient differs from zero, the metrics increase its value, indicating the difference and, thus, the error. Therefore, the case study shows that the metrics are able to capture differences in the shape of the function in any direction.

5.4 Case study 4: Analysis of errors in conditional statements

This case study explores the behaviour of the metrics when the probability of different branches in a conditional statement is incorrectly estimated.

A system has been implemented with only one task. Its parameters are shown in Table 7. The task has two execution times: 20 ms, which corresponds to a

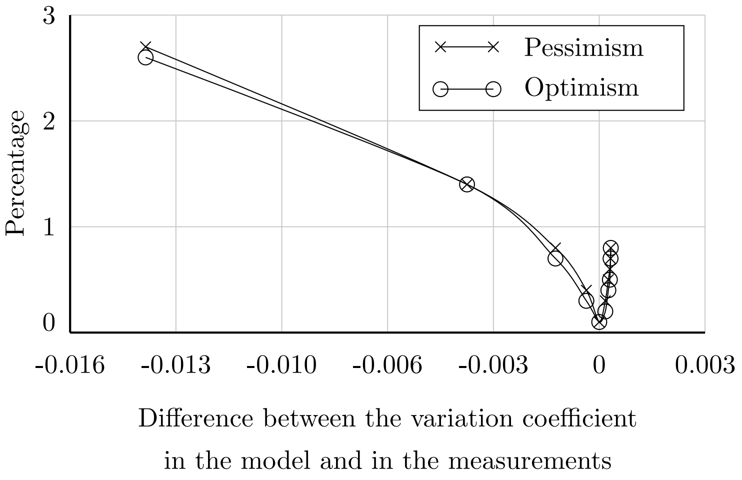


Figure 11: Evolution of the metrics vs variation in shape

branch of a conditional statement that will be dubbed "short branch"; and 100 ms, which correspond to the other branch of the conditional statement, which will be called "long branch".

Task	$T_i = D_i$	Φ_i	J_i	M_i	c	$f_{C_i}(c)$
τ_0	200	0	0.499504	0.1	20	0.5
					100	0.5

Table 7: Tasks in case study 4. All times are in ms

Both branches have the same probability in order to test what happens when the model estimates them incorrectly in any direction.

The system was measured for two hours, generating a trace that has been compared to eleven different models where the probability of the short branch was varied between 0 and 1 in steps of 0.1; consequently, the probability of the long branch had the same variation but from 1 to 0. The first model assigned a probability of 0 for the short branch and 1 for the long one, the second model assigned a probability of 0.1 to the short branch and 0.9 to the long one, and so on.

The results can be seen in Figure 12. When there is no error in the model (both branches have a probability of 0.5), pessimism and optimism are zero,

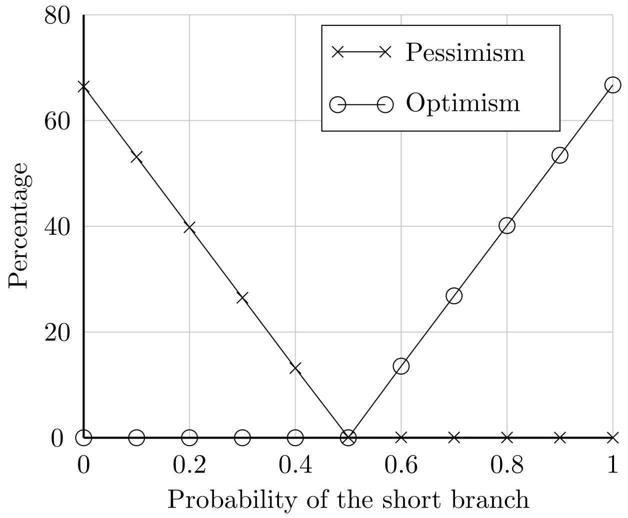


Figure 12: Behaviour of pessimism and optimism related to errors in the probability of a conditional statement

as expected. When the model incorrectly assigns less probability to the short branch (which can be seen moving to the right in the abscissas), pessimism grows linearly because the model is assigning less probability to shorter times and, thus, more probability to longer times. Similarly, when the probability of the short branch increases from 0.5 in the model, optimism grows linearly.

This case study shows that the metrics behave in a reasonable way for conditional statements: when the same error is introduced in different directions, the values in the metrics are symmetrical.

6 Conclusions and future work

This paper has presented the problem of debugging the timing behaviour of real-time systems when they are analyzed with stochastic techniques. As these techniques provide a more powerful analysis of real-time systems than traditional non-stochastic techniques, the problem addressed will be of great importance in the future.

After defining a timing error, two tools for debugging the timing behaviour of real-time systems have been presented. First, a monitoring tool which can measure POSIX systems in a portable way has been introduced, with its intrusiveness analyzed. Then, the visualization and analysis tool that, using the traces generated by the monitoring tool, carries out an analysis of the system and presents the results has been described.

The main contribution of the paper is the introduction of two metrics, named "pessimism" and "optimism", which deal with the problem of finding timing errors in the characterization of the computation time as a random variable. They are based on two new random variables, the optimistic and the pessimistic computation time, generated from information contained both in the model and in the measurements. As four case studies have shown, the metrics are useful in finding timing errors in stochastic systems.

The greatest limitation of the metrics is that they are heuristic and hence, do not provide a statistic confidence value depending on the sample size. This problem is very difficult to address because the samples are not independent. However, the analysis and visualization tool provides a window showing the evolution of the metrics that can be used to asses whether they have reached a steady state.

The metrics can also help in contexts different from debugging; for instance, instead of comparing a model distribution and a measurement distribution, they can be used to compare two different models or two different analysis techniques.

Future work will focus on extending the debugging approach to other parameters of stochastic systems such as the blocking time.

References

[Abeni and Buttazzo 2001] Abeni, L. and Buttazzo, G.: "Stochastic Analysis of a Reservation Based System." In Proc. of the 9th International Workshop on Parallel and Distributed Real-Time Systems.

[Chodrow et al. 1991] Chodrow, S.E., Jahanian, F., and Donner, M.: "Run-time monitoring of real-time systems." In Proc. of the 12th IEEE Real-Time Systems Symposium.

[Díaz et al. 2002] Díaz, J.L., García, D.F., Kim, K., Lee, C.G., Bello, L.L., López, J.M., Min, S.L., and Mirabella, O.: "Stochastic Analysis of Periodic Real-Time Systems in a Real-Time System." In Proc. of the 23rd IEEE Real-Time Systems Symposium.

[Díaz et al. 2004] Díaz, J.L., López, J.M., García, M., Campos, A.M., Kim, K., and Bello, L.L.: "Pessimism in the Stochastic Analysis of Real-Time Systems: Concept and Applications." In Proc. of the 25th IEEE Real-Time Systems Symposium.

[Dodd and Ravishankar 1992] Dodd, P.S. and Ravishankar, C.V.: "Monitoring and debugging distributed real-time programs." Softw. Pract. Exper., 22, 10, 863–877 (1992).

[Entrialgo et al. 2007] Entrialgo, J., García, J., Díaz, J.L., and García, D.F.: "Stochastic Metrics for Debugging the Timing Behaviour of Real-Time Systems." In Proc. of the 13th IEEE Real Time and Embedded Technology and Applications Symposium. IEEE Computer Society.

[Entrialgo et al. 2003] Entrialgo, J., García, J., and García, D.F.: "Measurement-Based Analysis of Real-Time POSIX Applications." In Proc. of the 7th IASTED International Conference on Software Engineering and Applications.

[Gergeleit and Nett 1999] Gergeleit, M. and Nett, E.: "JewelNT: Monitoring of Distributed Real-Time Application on Windows NT." In Proc. of the 3rd Annual IASTED Intern. Conference on Software Engineering and Applications.

[IEEE 2004] IEEE: 1003.1, 2004 Edition, Standard for Information Technology - Portable Operating System Interface (POSIX), System Interfaces. The Institute of Electrical and Electronics Engineers (2004).

[Kim et al. 2005] Kim, K., Díaz, J.L., Bello, L.L., López, J.M., Lee, C.G., and Min, S.L.: "An Exact Stochastic Analysis of Priority-Driven Periodic Real-Time Systems and Its Approximations." IEEE Transactions on Computers, 54, 11, 1460–1466 (2005).

[LeBlanc and Mellor-Crummey 1987] LeBlanc, T.J. and Mellor-Crummey, J.M.: "Debugging parallel programs with instant replay." IEEE Transactions on Computers, 36, 4, 471–482 (1987).

[Lehoczky 1990] Lehoczky, J.P.: "Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines." In Proc. of the 11th IEEE Real-Time Systems Symposium.

[Liu and Layland 1973] Liu, L. and Layland, J.: "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment." Journal of ACM, 20, 1, 46–61 (1973).

[López et al. 2008] López, J.M., Díaz, J.L., Entrialgo, J., and García, D.F.: "Stochastic Analysis of Real-Time Systems under Preemptive Priority-Driven Scheduling." Real-Time Systems, 40, 2, 180–207 (2008).

[Manolache et al. 2001] Manolache, S., Eles, P., and Peng, Z.: "Memory and Time-Efficient Schedulability Analysis of Task Sets with Stochastic Execution Times." In Proc. of the 13th Euromicro Conference on Real-Time Systems.

[Mok and Liu 1997] Mok, A.K. and Liu, G.: "Efficient Run-Time Monitoring of Timing Constraints." In Proc. of the 3rd IEEE Real-Time Technology and Applications Symposium. IEEE Computer Society.

[Petters 2002] Petters, S.M.: Worst Case Execution Time Estimation for Advanced Processor Architectures. Ph.D. thesis, Institute for Real–Time Computer Systems, Technische Universität München, Munich, Germany (September 2002).

[Raju et al. 1992] Raju, S.C.V., Rajkumar, R., and Jahanian, F.: "Monitoring Timing Constraints in Distributed Real-Time Systems." In Proc. of the 13rd IEEE Real-Time Systems Symposium.

[Stewart and Arora 2003] Stewart, D.B. and Arora, G.: "A Tool for Analyzing and Fine Tuning the Real-Time Properties of an Embedded System." IEEE Transactions on Software Engineering, 29, 4, 311–326 (2003).

[Terrasa and Bernat 2003] Terrasa, A. and Bernat, G.: "Extracting Temporal Properties from Real-Time Systems by Automatic Tracing Analysis." In 9th Intl Conf. on Real-Time and Embedded Computing Systems and Applications.

[Thane et al. 2003] Thane, H., Sundmark, D., Huselius, J.G., and Pettersson, A.: "Replay Debugging of Real-Time Systems Using Time Machines." In Proc. of the International Parallel and Distributed Processing Symposium, presented at the First International Workshop on Parallel and Distributed Systems: Testing and Debugging.

[Timmerman et al. 1993] Timmerman, M., Gielen, F., and Lambrix, P.: "High level tools for the debugging of real-time multiprocessor systems." SIGPLAN Not., 28, 12, 151–157 (1993).

[Tindell et al. 1994] Tindell, K., Burns, A., and Wellings, A.J.: "An Extendible Approach for Analyzing Fixed Priority Hard Real-Time Tasks." Real-Time Systems, 6, 2, 133–151 (1994).

[Tokuda et al. 1988] Tokuda, H., Kotera, M., and Mercer, C.E.: "A real-time monitor for a distributed real-time operating system." In Proc. of the 1988 ACM SIGPLAN and SIGOPS Workshop on Parallel and Distributed Debugging. ACM Press.

[Tsai et al. 1996] Tsai, J.J.P., Bi, Y., Yang, S.J.H., and Smith, R.A.W.: Distributed real-time systems: monitoring, visualization, debugging, and analysis. John Wiley & Sons, Inc., New York, NY, USA (1996).

[Tsai and Yang 1995] Tsai, J.J.P. and Yang, S.J.H. (eds.): Monitoring and debugging of distributed real-time systems. IEEE Computer Society Press, Los Alamitos,

CA, USA (1995).
[Wilner 1995] Wilner, D.: "WindView: a tool for understanding real-time embedded software through system vizualization." In Proc. of the ACM SIGPLAN 1995 workshop on Languages, compilers, & tools for real-time systems. ACM Press.
[Yaghmour 2000] Yaghmour, K.: "Monitoring and Analyzing RTAI System Behavior Using the Linux Trace Toolkit." In Proc. of the 2nd Real-Time Linux Workshop.