Stochastic Metrics for Debugging the Timing Behaviour of Real-Time Systems

Joaquín Entrialgo, Javier García, José Luis Díaz, Daniel Fernando García
Departamento de Informática
Campus de Viesques, 33204, Gijón, Spain
{joaquin,javier,jldiaz,dfgarcia}@uniovi.es

Abstract

Stochastic analysis techniques for real-time systems model the execution time of tasks as random variables. These techniques constitute a very powerful tool to study the 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. In this paper, a set of tools to measure, analyze and visualize traces of real-time systems is presented. These tools are driven by stochastic models. In order to find bugs in the timing behaviour of the system, 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 three case studies.

1 Introduction

In order to prove that a real-time system fulfills all of its timing constraints, various methods have been proposed. Traditional techniques, such as the processor utilization analysis [8, 7] and response time analysis [13], use a model of the tasks of the system where the execution time of the tasks is represented by the 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 results in too pessimistic WCETs, 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 [1, 9, 3]. These techniques analyse the timing properties of the system based on a stochastic model of its timing properties. Usually, the execution time of each task in the system, instead of being modeled just with the worst case execution time, is modeled 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. However, any mistake in the model or in the application of the techniques can lead to an error in the system. Thus, they can not be the only technique guaranteeing the system performance — the system must also be measured in order to find errors.

One important step to remove the errors from a system, i.e., to debug it, is finding where the errors lie. In the context of timing analysis, this 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

This work is related to two fields: stochastic analysis techniques of real-time systems and monitoring systems for debugging temporal behaviour of real-time systems.

Stochastic analysis techniques are used as a base for analysing of the measurements, so no contribution is made to the techniques themselves; however one of them must be

chosen. The one presented in [6] has been chosen because it provides the most exact analysis without needing strong restrictions, as shown in [2].

Much work has been done dealing specifically with monitoring real-time systems. A compilation of the most significant work can be found in [15] and [14]. None of these papers deal with stochastic analysis techniques and most of them do not relate their measurements with any kind of analysis model, which are the main contributions of the work presented here. More recent research deals with some of these topics, but not all. [10] measures blocks of code in order to feed stochastic models, but is not aimed at debugging. [12] shows how to obtain metrics from a trace, but does not take stochastic techniques into account. [11] aims to find bugs comparing measurements and models, but uses a non-stochastic model, with a very restricted implementation.

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 debugging strategy proposed 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 is finding errors and pinpointing 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.

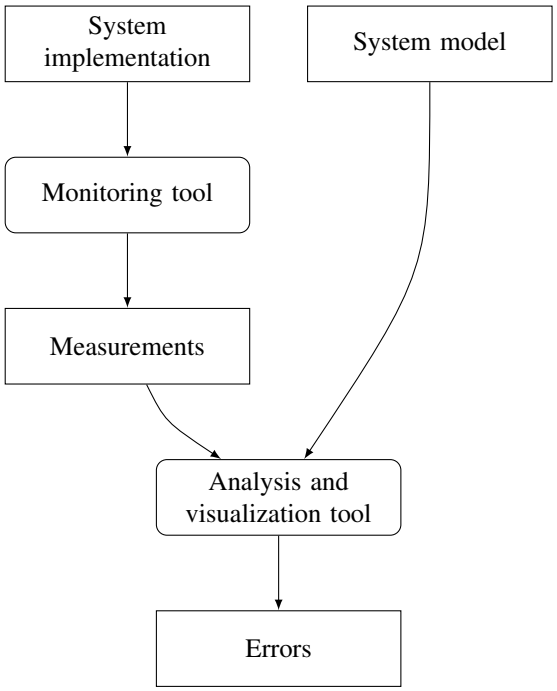

Figure 1. Overview of the debugging strategy

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

The system model used is based on the model presented in [6]. The system is composed of a set of n independent periodic tasks $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_n\}$. With no 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, 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

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

Using the equations in [2], 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 calculation of a job random response time can be found in [6].

3.3 System Implementation

An implementation based on POSIX [5] 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 basic objectives, so it works at source level, as this is the only level standardized by POSIX.

The monitoring tool instruments the source code by adding some 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 monitoring tool does not introduce more threads in the system. Its instructions are executed in the thread where events are gathered. In order to avoid sharing a resource between all the threads, which would change significantly the system model, there is an event buffer for each thread.

The maximum error in a timestamp depends on the precision of the clock that can be accessed with POSIX functions. In the test platform described in Table 2, it is 0.499504 milliseconds.

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. A more detailed description of the monitoring system, including a discussion on intrusiveness, can be found in [4].

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.

First, the tool shows a summary of the system state in the "Metrics Windows", which provides a series of metrics for each task, based on the system model, the measurements and a comparison of both. One of the main areas of the window shows a list of problems found, which the user can further investigate with the help of the metrics and by using supplementary windows with other graphs, such as Gantt diagrams or probability functions.

In order to find errors, the tool compares the values in the model and values obtained from the trace. How this comparison is carried out for stochastic values is discussed in the following section.

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 [3], 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 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 by $\mathcal{X} \succcurlyeq \mathcal{Y}$ 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 $\mathcal{Y}$, so $\mathcal{X}$ 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 be just 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.

In order to show this problem, an example will be introduced. Let us consider the model presented in Table 1.

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 1. Example model. All times are in ms

Figure 2 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 measurements CDF and the model CDF, because the real CDF is unknown. In this example, the model is not more pessimistic than the measurements CDF — there are parts of the model curve that go above the measurement curve.

What this example shows is that the decision as to whether there is an error can not be made by simply assessing if the model CDF is always below the measurements CDF: the sampling error can make the measurements CDF 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 measurements CDF and the real CDF should be very close; thus, in a system without timing errors there should be few intervals where the measurements CDF 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.

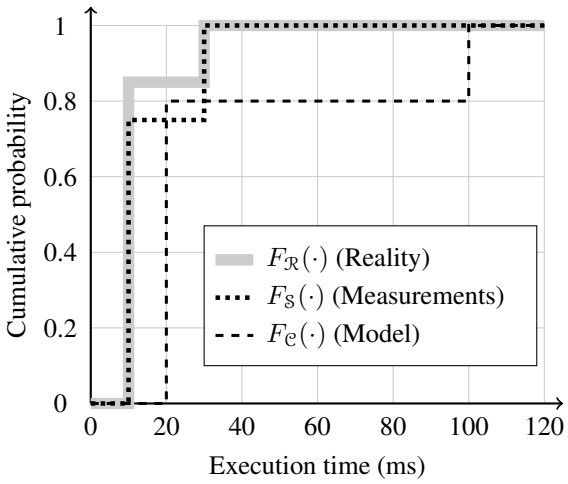

Figure 2. Probability distribution example

Instead of a statistical test, in this work we propose using heuristic metrics. The goals pursued in defining this 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 measurements CDF 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 have to 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** $\mathcal{C}^{\boldsymbol{O}}$ is defined as the random variable which has the following CDF:*

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

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

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

Figures 3 and 4 show a graphical interpretation of the CDF of these variables by means of an example. As seen in Figure 3, the CDF of the optimistic execution time is the curve that always follows the highest of the measurements CDF and the model CDF. Similarly, as seen in Figure 4, the CDF of the pessimistic execution time is the curve that follows the lowest of the measurements CDF and the model CDF.

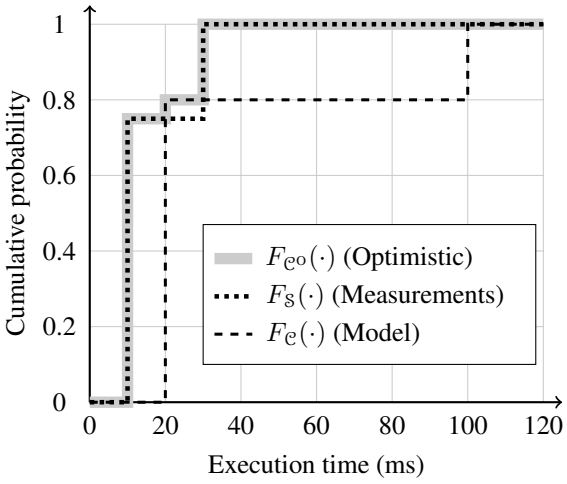

Figure 3. Optimistic execution time example

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 will be used. By using the optimistic execution time, there will be no compensation of optimism with pessimism. In a similar way, pessimism will be 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:*

$$O = \frac{\bar{S} - \bar{C}^O}{\bar{S}} \quad (4)$$

Definition 6 ***Pessimism*** *is defined as:*

$$P = \frac{\bar{C}^P - \bar{S}}{\bar{S}} \quad (5)$$

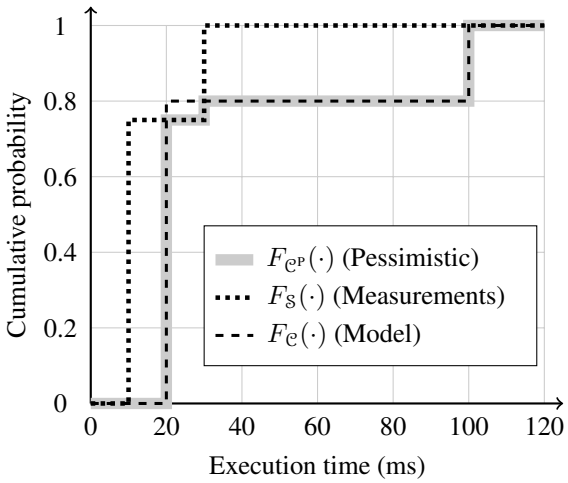

Figure 4. Pessimistic execution time example

In the previous equations, $\bar{S}$, $\bar{C}^O$ y $\bar{C}^P$ are the mean of the random variables $\mathcal{S}$, $\mathcal{C}^O$ and $\mathcal{C}^P$ 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 measurements CDF. 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 three case studies. All of the case studies have been implemented in a platform with the characteristics shown in Table 2. The platform uses QNX as its operating system, which is one of the main real-time operating systems and follows the POSIX standard.

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 2. 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 2. Table 3 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. As seen in Table 4, acccording to the analysis of the model with the techniques presented in [2], 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 3. Model for the case study 1. All times are in ms

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

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 4. Model analysis results for the case study 1

The implementation has been measured for two hours using the monitor described in Section 3.4. After opening the corresponding traces with the analysis and visualization tool described in Section 3.5, the "Metrics Window" shown in Figure 5 presents a summary of the system behaviour.

The Metrics Window is divided in 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, from a huge amount of information, the relevant data. An automatic analysis searching for bugs is carried out and, as a result, the cell with data that pinpoint errors are highlighted in the table, and messages guiding the analyst to further inspection into 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 the task τ_1. The missed deadline ratio for task τ_3 is more than double of its specified maximum allowable ratio of deadline misses (0.1, as can be seen in Table 5.1).

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 the ten percent 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 6 shows this window for task τ_1.

In this figure the error that was introduced is evident: the two probabilities in task τ_1 are interchanged. Thus, the

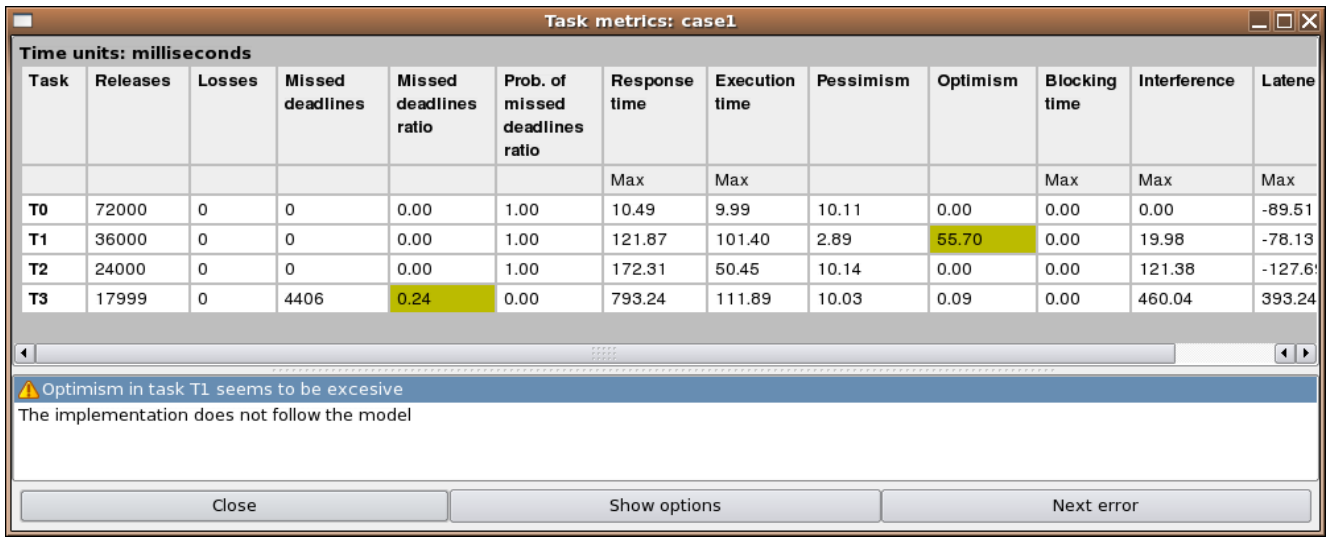

Figure 5. Metrics of Case Study 1

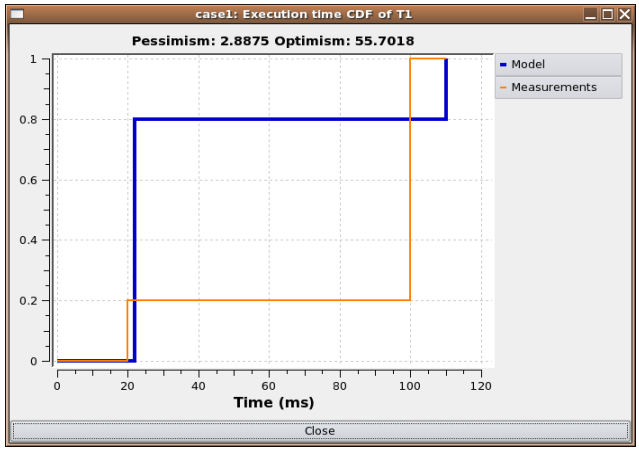

Figure 6. CDFs of task τ_1 of Case Study 1

tool has helped in finding an error otherwise very difficult to locate without its assistance.

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 5, 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 5

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 5. Model for the case study 2. All times are in ms

A system with these parameters was measured for two hours. Figure 7 shows a window from the analysis and visualization tool with the evolution of pessimism and optimism for task τ_0 computed as 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 plat-

form 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.

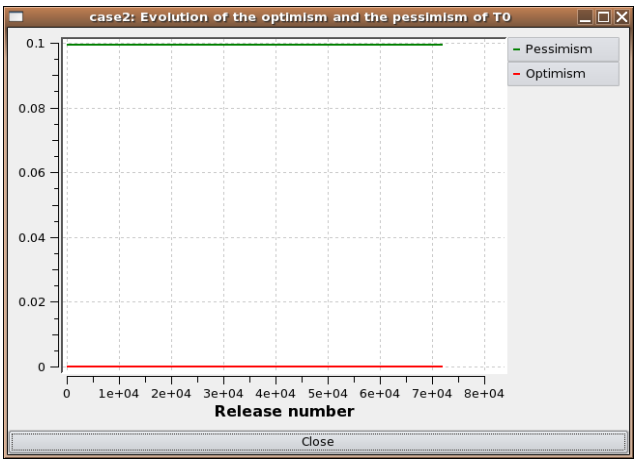

Figure 7. Evolution of the metrics for τ_0 of Case Study 2

Figure 8 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 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 include 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

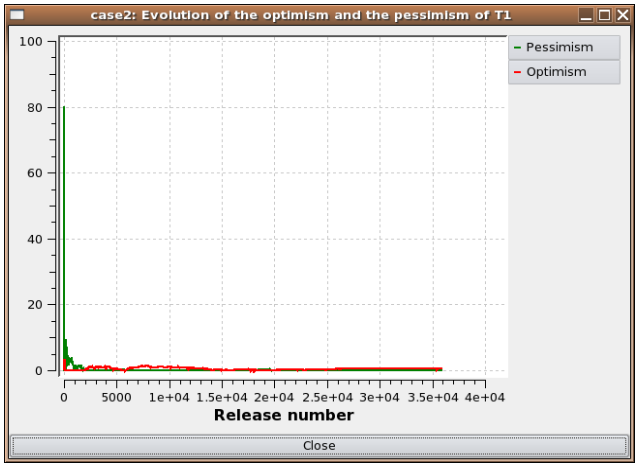

Figure 8. Evolution of the metrics for τ_1 of Case Study 2

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.

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 9 shows the probability function of the original beta function ($\alpha = 2$, $beta = 5$) and of two additional models. As can be seen, the shape of the functions varies; however, their means are the same.

In order to test the metrics, the ten different models have been analyzed with the analysis and visualization tool using the same trace obtained with the original beta function. Figure 10 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

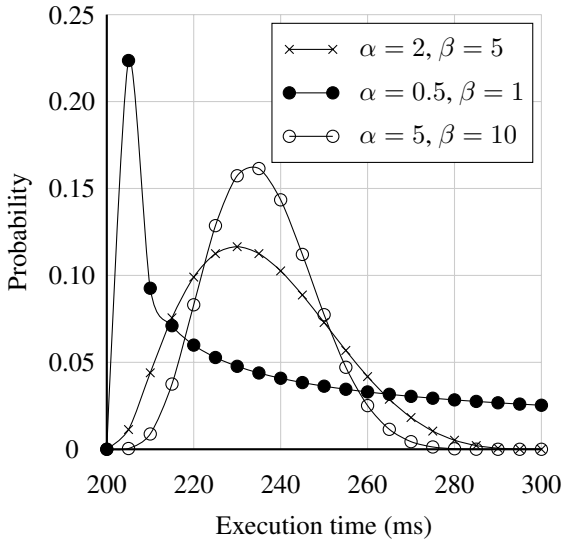

Figure 9. Beta functions used in case study 3

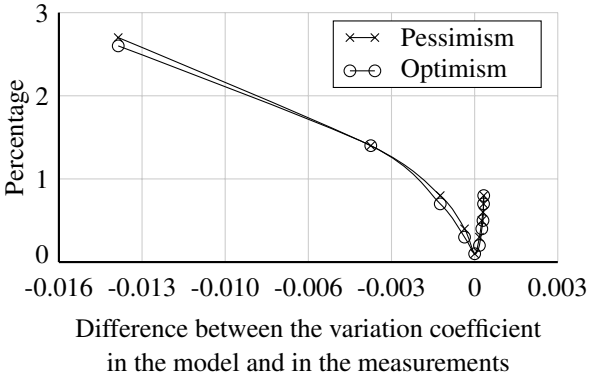

Figure 10. Metrics evolution vs variation in shape

respect to the mean has a significant impact on the metrics. As the metrics differ from zero whenever the variation coefficient also differs from zero, the case study shows that the metrics are able to capture differences in the shape of the function in any direction.

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 what is considered a timing error, a set of tools to measure, analyze and visualize the timing behaviour of real-time systems has been proposed. 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 three 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 help also 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

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

[2] J. L. Díaz, D. F. García, K. Kim, C.-G. Lee, L. L. Bello, J. M. López, S. L. Min, and O. Mirabella. Stochastic Analysis of Periodic Real-Time Systems in a Real-Time System. In *Proc. of the 23rd IEEE Real-Time Systems Symposium*, pages 289–300, Austin, Texas, Dec. 2002.

[3] J. L. Díaz, J. M. López, M. García, A. M. Campos, K. Kim, and L. L. Bello. Pessimism in the Stochastic Analysis of Real-Time Systems: Concept and Applications. In *Proc. of the 25rd IEEE Real-Time Systems Symposium*, pages 197–207, Lisbon, Portugal, Dec. 2004.

[4] J. Entrialgo, J. García, and D. F. García. Measurement-based analysis of real-time posix applications. In *Proc. of the 7th IASTED International Conference on Software Engineering and Applications*, Marina del Rey, CA, USA, Nov. 2003.

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

[6] K. Kim, J. L. Díaz, L. L. Bello, J. M. López, C. G. Lee, and S. L. Min. An exact stochastic analysis of priority-driven periodic real-time systems and its approximations. *IEEE Transactions on Computers*, 54(11), Nov. 2005.

[7] J. P. Lehoczky. Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines. In *Proc. of the 11th IEEE Real-Time Systems Symposium*, pages 201–209, Dec. 1990.

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

[9] S. Manolache, P. Eles, and Z. Peng. Memory and Time-Efficient Schedulability Analysis of Task Sets with Stochastic Execution Times. In *Proc. of the 13th Euromicro Conference on Real-Time Systems*, pages 19–26, Jun. 2001.

[10] S. M. Petters. *Worst Case Execution Time Estimation for Advanced Processor Architectures*. PhD thesis, Institute for Real–Time Computer Systems, Technische Universität München, Munich, Germany, Sept. 2002.

[11] D. B. Stewart and G. Arora. A tool for analyzing and fine tuning the real-time properties of an embedded system. *IEEE Trans. Softw. Eng.*, 29(4):311–326, 2003.

[12] A. Terrasa and G. Bernat. Extracting temporal properties from real-time systems by automatic tracing analysis. In *9th Intl Conf. on Real-Time and Embedded Computing Systems and Applications*, 2003.

[13] K. Tindell, A. Burns, and A. J. Wellings. An Extendible Approach for Analyzing Fixed Priority Hard Real-Time Tasks. *Real-Time Systems*, 6:133–151, 1994.

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

[15] J. J. P. Tsai and S. J. H. Yang, editors. *Monitoring and debugging of distributed real-time systems*. IEEE Computer Society Press, Los Alamitos, CA, USA, 1995.