Modelling Fault Dependencies when Execution Time Budgets are Exceeded

David Griffin
University of York
david.griffin@york.ac.uk

Benjamin Lesage
University of York
benjamin.lesage@york.ac.uk

Iain Bate
University of York
iain.bate@york.ac.uk

Frank Soboczenski
University of York
frank.soboczenski@york.ac.uk

Robert I. Davis
University of York and Inria Paris-Rocquencourt
rob.davis@york.ac.uk

ABSTRACT

Given that real-time systems are specified to a degree of confidence, budget overruns should be expected to occur in a system at some point. When a budget overrun occurs, it is necessary to understand how long such a state persists, in order to determine if the fault tolerance of the system is adequate to handle the problem. However, given the rarity of budget overruns in testing, it cannot be assumed that sufficient data will be available to build an accurate model. Hence this paper presents a new application of Markov Chain based modelling techniques combined with forecasting techniques to determine an appropriate fault model, using Lossy Compression to fit the model to the available data. In addition, a new algorithm, *DepET*, for generating job execution times with dependencies is given for use in task simulators.

1. INTRODUCTION

Real-time systems [5] are characterised by a need for correctness of their temporal properties, in addition to their logical properties. As such, a fault in a system can manifest itself through overrunning the amount of time allocated to a task to execute i.e. either the anticipated Worst Case Execution Time (WCET) or Worst Case Response Time (WCRT) of the task being exceeded. As ruling out such temporal faults completely is normally an intractable problem, alternative arguments must be presented. In practice, this means that a system will be engineered to withstand a certain level of faults, defined as the *fault tolerance* of the system. Provided that the faults the system encounters do not exceed its ability to deal with them, the system will be able to continue to function [15]. For example, aircraft and automotive Electronic Engine Controllers (EEC) are designed to have a fallback by outputting the last calculated value in the case of a budget overrun; this approach is not restricted to EECs [15, 14]. While this strategy may allow the system to continue operating safely if budget overruns are rare events, a burst of faults would present a difficult argument for the safety case.

RTNS 2015, November 04 - 06, 2015, Lille, France
Copyright 2015 ACM 978-1-4503-3591-1/15/11 ...$15.00.
http://dx.doi.org/10.1145/2834848.2834870.

The reliability of a system is typically measured by its acceptable failure rate; provided that this failure rate is not exceeded, the system will be able to continue to operate. Given a system comprised of a set of *tasks*, an execution budget failure is caused when an instance of a task, a *job*, exceeds its WCET budget. The rates of such failures are typically mandated by an appropriate standard and a WCET budget selected such that the failure rate is satisfied [22]). Recent techniques include measurement based probabilistic timing analysis (MBPTA) [9], where statistical methods are used to predict the rate of execution time budget overruns. However, while normally presented as a single figure, the probability of a fault occurring does not convey all the relevant information. For example, consider a task that out of 10,000 executions, observes 30 consecutive faults. Over all the executions, the failure rate is 0.03%, which appears low. However, if it is restricted to the last 100 executions, the failure rate becomes 30%. Indeed, in the last 10 executions a 100% failure rate is observed, which is clearly undesirable.

This leads to the conclusion that the term "failure rate", as used in MBPTA literature [9], actually refers to the probability of an initial fault. Once this initial fault has occurred, the given failure rate does not necessarily remain valid unless faults occur independently, which is a common assumption in previous fault models [4]. In addition, the related concept of weakly hard real time systems [2] also makes the assumption that faults occur independently. Research has also been conducted on the assumption that due to the rarity of budget overruns, consecutive budget overruns are a rare occurrence [19]. However, if the inputs to the system exhibit any kind of dependencies, one would expect that dependencies could also manifest in system execution times, therefore causing dependent faults. An example of this can also be observed in EECs; given the rate of change in the physical system providing input (the engine, accelerator position, etc.), the inputs to the task will not change substantially between samples and therefore similar paths through the program are likely to be exercised, leading to a burst of faults.

Therefore it becomes necessary to consider the behaviour of a system once a fault has been observed. This is especially relevant in systems where the inputs to tasks change by a limited amount between jobs, such as the aforementioned EEC example. Hence if a particular job exceeds its WCET due to the computation required to process its inputs, then the next job, expected to have similar inputs, is expected to be at an increased risk of a budget overrun when compared to normal behaviour.

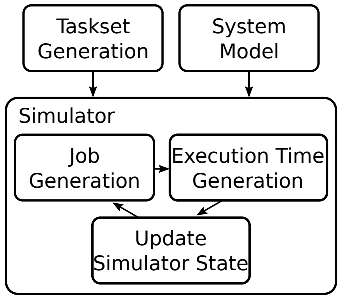

Figure 1: Workflow of a Scheduling Simulator

However, the problem encountered in attempting to model the behaviour of a system after a fault is that faults are, by definition, rare events. It could be reasonably expected that during testing, not enough faults are observed and therefore it is impossible to gather sufficient information about the system's behaviour after a fault.

This problem is demonstrated by approaches seen in current state of the art scheduling simulators. As illustrated in Figure 1, a scheduling simulator given a taskset and system model, simulates how jobs will behave. For example, the SimSo simulator [6] is capable of considering a variety of execution time models for jobs. However, in these models the base execution times are produced by using sampling from a normal distribution, with some additional factors allowing limited dependencies for the cases of premption or task migration. Even removing this source of dependencies, it is still unlikely that a real system will experience jobs where the base execution times are independent, due to dependencies in input to the system. Therefore, this assumption introduces an inaccuracy in the simulator, and an improved execution time generator is therefore desirable.

This inaccuracy is highlighted in mixed criticality systems (MCS) [4] where the system is specified at multiple levels. In the simplest case, the system has two levels, *low criticality*, where budget overruns do not usually occur, and *high criticality*, a reduced functionality mode with increased execution time budgets for the remaining tasks such that budget overruns are *guaranteed* not to occur [4]. In the case that a WCET budget overrun occurs in low criticality mode, the system transitions to high criticality mode to guarantee no further budget overruns. However, to do this, low criticality functionality is sacrificed. Given that current simulators do not provide for dependencies for when budget overruns occur, this presents a challenge for evaluating the effectiveness of MCS algorithms.

As Markov Chain modelling is a widely used technique with broad applicability, this paper presents an algorithm which utilises Markov Chains to construct a model for job execution times exceeding a given threshold, which is capable of modelling faults caused by exceeding a WCET budget. The novel step of this algorithm is the use of Lossy Compression to guarantee statistical confidence in the model, as well as the use of forecasting methods to counteract the rarity of observing actual WCET budget exceedances. To the authors knowledge, this is the first application of forecasting techniques to the job execution time generation problem, and enables a prediction of the behaviour of the system under extreme circumstances.

In addition, this paper presents a new job execution time generation algorithm, *DepET*, which utilises the forecast models of job execution time. *DepET* uses these models to target an overall distribution while also giving temporal dependencies between the execution times of jobs. When used in a task simulator, *DepET* distinguishes itself by presenting more realistic job execution times, and therefore removes a source of inaccuracy commonly found in task simulators.

2. RELATED WORK

While little research has been conducted on the occurrence of faults due to budget overruns in a system, techniques have been employed in predicting the existence of functional faults in software by Fenton et al. [12]. Fenton et al. used a set of training data to create a Bayesian Network which in turn could predict, with reasonable accuracy, the number of faults to be expected in other pieces of software based on simple observations of the software (e.g. lines of code, size of development team etc.). However, Bayesian Networks are designed to assign a probability to a particular outcome given various inputs, which does not match the class of problems that a fault model would seek to solve, due to the fact that a fault model must attempt to produce realistic data on the nature of faults. Such data may in turn be repetitive, which is something that cannot be expressed in a Bayesian Network. Hence, the related construct of Markov Chains [20] is employed instead.

A Markov Chain is a conditional probability model that satisfies the Markov Property: "Given the present state, future states are independent of past states". This property provides a simplification of real systems in that it limits the effect of dependencies between states, but provides a compromise which allows a limited amount of information to influence dependencies. Unlike alternative systems, the Markov Property that a Markov Chain adheres to ensures that the resulting model is sufficiently simple that only a limited amount of data is required to train it.

In order to find information on events which have not been observed, Forecasting methods [13] are employed. Forecasting allows a *forecast model* to be populated with observed data, followed by the identification of a trend or pattern within the model, and therefore an expectation of future unobserved events. Hence forecasting methods are an appropriate technique to attempt to predict the behaviour of faults given their low frequency in observed data; patterns are identified in observed data and then extrapolated to provide a prediction of unobserved data. While forecasting is not typically employed on real-time systems, there are limited methods available in order to predict unobserved data. In particular, this paper focuses on the technique of *extrapolation*: the identification of a pattern or trend in observed data which is then used to predict the behaviour of unobserved phenomena. It should also be noted that these techniques allow underlying factors of the system being modelled (e.g. details of the tasks or the computer architecture) to be abstracted; during evaluation, it is possible to determine the accuracy of the forecast model relative to the actual system. This demonstrates that the model is accurate, and hence the potential downside of not using detailed information on the target system is mitigated.

Lossy Compression, previously used on models of real time systems by Griffin et al., has been applied to static analyses of PLRU caches [16, 18] and random replacement caches [17]. Given that Lossy Compression is capable of simplifying the models used in static analysis without discarding useful information, the same principles can be

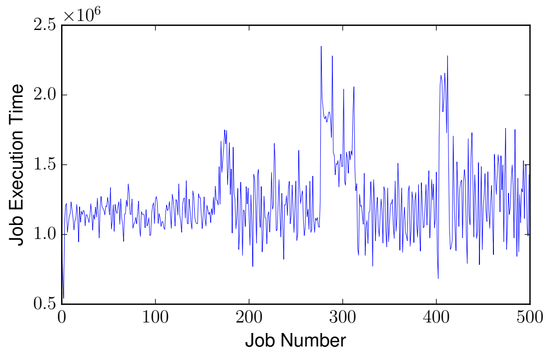

Figure 2: Instructions taken for FFmpeg 2.4.3 decoding video frames of *Return of the Kung Fu Dragon* (MPEG4)

used to simplify a model populated by data from measurements. Specifically, this is used to take a generic, highly detailed model, and simplify the model until the observations available are sufficient to to populate the aspects of the model with enough data to give statistical confidence.

3. CONSECUTIVE EXECUTION TIMES AND BUDGET OVERRUNS

As previously stated, during the execution of a job it is expected that the inputs to that job will not change significantly when compared to previous invocations of the task. While there are examples of systems which break this rule (e.g. random number generators), for the majority of systems this is a realistic assumption. For example, the velocity of an aircraft does not vary wildly (as governed by the laws of physics), and indeed will have a maximum rate of change. Similarly, in video decoding the content of a video frame is likely to be similar to the previous video frame, and therefore takes a similar amount of effort to encode. To extend this analogy to recent encoding technologies which utilise different types of frame (normally I-frames and P-frames), the input for each type of frame is similar to the previous instance of that type of frame. Assuming that inputs are similar, then it is reasonable to assume that the amount of effort that the job must expend processing those inputs is likely to be similar.

This assumption results in the kind of distribution of execution times that is seen in Figure 2. Figure 2 gives the number instructions taken for decoding the P-frames of an XVID video using FFmpeg [7], extracted by running the program under Callgrind [8]. This data source was chosen due to the fact that it is an easily obtainable example of a system which is commonly deployed, and for which data can be easily extracted using precise instrumentation. In addition, due to the use of the number of instructions taken rather than execution time being used to model task length, many of the specific details of the CPU being used become irrelevant. While the number of instructions taken is not entirely analogous to the execution time of the job in question, it does provide a reproducible result which captures enough of the actual behaviour to provide a useful data source. Throughout this paper, data was extracted by decoding the film *Return of the Kung Fu Dragon*[1].

Further, FFmpeg is a highly complex piece of software which exhibits real-time properties (in that decode times appear to be bounded), and has no shortage of valid, real, input data; this is in contrast to other public domain benchmarks (EEMBC, MDH), where real input data is limited. Given the shape of the graph, it is clear that a good indicator of the execution time of the next job is the execution time of the previous job. The ratio of these quantities is plotted in Figure 3, which show that the execution time of a job is most commonly within 20% of the execution time of the previous job, and rarely outside of a 30% margin.

[1]https://archive.org/details/Return_of_the_Kung_Fu_Dragon

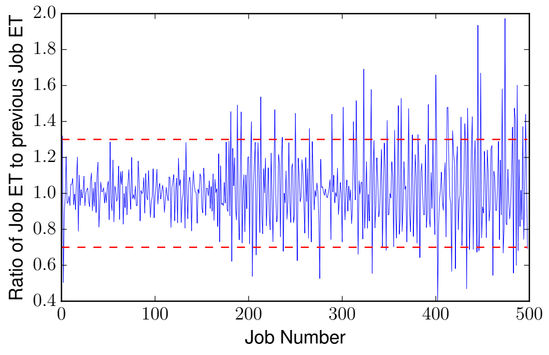

Figure 3: Correlation between the execution time of a job and the execution time of the previous job

Duration of Fault	Number of instances
$0-5$	12685
$6-10$	1082
$11-15$	362
$16-25$	281
$26-50$	85
$51-75$	63
> 75	50

Figure 4: Duration of consecutive budget overruns for FFmpeg decoding video frames

Given this strong correlation, it confirms the position that for any given execution time threshold, execution times exceeding the threshold tend to be observed in consecutive groups. Hence, if one were to set such a threshold for budget overruns, budget overruns also tend to be observed in groups as seen in Figure 4. In this paper, the specific quantity modelled is the duration of budget overruns, defined as once a budget overrun has occurred, the number of additional task invocations before normal operation is resumed. This was chosen instead of modelling the magnitude of the overruns (the amount of time spent on consecutive jobs not budgeted for) as many task schedulers will take action when an overrun occurs (e.g. killing the job or switching criticality mode).

Unfortunately, the number of budget overruns observed during testing is unlikely to be sufficient to construct an accurate model of the system's behaviour after a budget overrun, at any reasonable confidence. For example, to obtain a statistically significant sample of 1000 job overruns which occur with a probability of 10^{-6}, then one would need to execute the system between 10^6 and 10^9 times depending on the degree of dependence between overruns. However, referring back to Figure 2, one can see that budget overruns are likely to occur around the peaks of consecutive computationally expensive jobs; this is due to the previous observation that a strong indicator of the execution time of a job, is the execution time of the

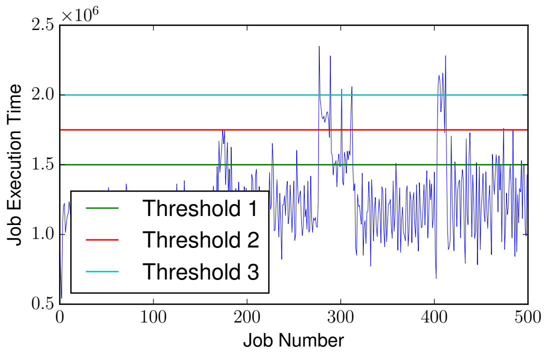

Figure 5: Setting thresholds such that sufficient data is available to construct a model

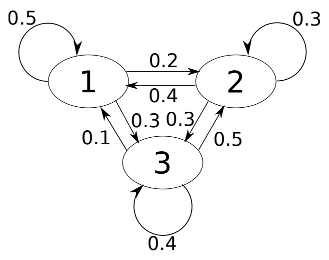

Figure 6: An example of a Markov Chain, where the states are annotated with 1, 2, 3 and state transitions occur with the given probabilities.

previous job of the same task. As illustrated in Figure 5, thresholds can be set such that these jobs are observed in high numbers. Hence, one could use these lower thresholds to capture information on the computationally expensive jobs, and then by varying the threshold, extrapolate the information to the cases where budget overruns occur.

Specifically, this paper hypothesises that information about the behaviour of runs of jobs after the system has experienced a fault due to a budget overrun (an event not seen in testing with any high confidence) can be inferred by collecting data on computationally expensive runs of jobs, and using this information to extrapolate the behaviour of the system after a budget overrun. In the evaluation (Section 7), this hypothesis is tested by comparing the forecast model of the system with actual observations.

In order to accomplish this, it is necessary to select an appropriate model for the system. The model chosen is a Markov Chain based model, which is outlined in the next section.

4. MARKOV CHAIN MODELS

Markov Chains [20] are a mathematical model which allows conditional probabilities based only on the current state; this ensures the Markov Property, "Given the Current State, the Past and Future are Independent" holds [20]. Markov chains are useful for modelling a number of systems, especially when the actual model is unknown; this is primarily due to their flexibility in capturing a limited amount of conditional behaviour.

As stated in the previous section, the overall strategy is to determine a model of behaviour after a budget overrun, by extrapolating from the behaviour of the system when the overrun threshold is lowered to a level at which the system can be tested. On it's own, a model of the duration of a budget overrun is not capable of providing a job execution time generator; the additional information required for this is laid out in the *DepET* algorithm in Section 4. As the model is only of the duration of an budget overrun, this extra information includes the probability of a budget overrun which will cause the exceedance model to be sampled from.

At its simplest, a Markov Chain is composed of a number of states (which may be annotated with additional information), each containing a probability table which gives the probabilities for which state is advanced to next. An example of a Markov Chain is given in Figure 6. In this case, the Markov Chain is evaluated by considering the probability table of the current state α_s. The next state, α_{s+1} is chosen in accordance with the probability table contained within the current state α_s. Repeated sampling allows the limited amount of history contained within the current state to inform the probabilities of the next state, as required. In order to extrapolate the behaviour of budget overruns, it is necessary to use a fixed structure for the Markov Chain. This ensures that the values contained in the probability tables are of the same type and therefore comparable. In order to accomplish this, each node of the Markov Chain corresponds to a range of durations of consecutive budget overruns; this allows sufficient data to be supplied to each node such that there is confidence in the results.

Ideally, the Markov Chain would comprise nodes where the state represents the number of consecutive budget overruns, and the probability table gives the probabilities for the next state. This would enable the model to evaluate the likelihood of the size of the next set of consecutive budget overruns in the context of the current set of budget overruns. This capability is important to distinguish different patterns of budget overruns, such as cases where a large number of budget overruns are followed by a short period with no budget overruns, but more budget overruns are likely to occur because the computational load has not disappeared, as illustrated in Figure 5. In order to accomplish this, the input data is then categorised appropriately and this information used to inform the probability tables in the Markov Chain. However, the amount of data obtainable through testing is likely to fall well short of being able to sufficiently populate such a model, due to the number of states that the model contains. In order to address this issue, the next section introduces Lossy Compression which is used to modify the model such that the evidence is capable of supporting it.

5. COMPRESSING THE MARKOV CHAIN MODEL

To compensate for a lack of data, Lossy Compression [18] is applied to the model. Lossy compression states that in order to compress a model, one should first list the types of information contained in the model, evaluate their importance to the desired result, and discard or approximate information which is of low importance to the overall result. By doing so, the least useful information is lost first, and hence the effects on the accuracy of the results are minimised. A common example of Lossy Compression is image compression (e.g. JPEG), where details of the image are ranked based on how perceptible they are to the human eye, with the least perceptible elements being compressed. Further, while previous applications of Lossy Compression were to Static Analysis [18], there is an additional benefit to applying Lossy Compression to a Measurement Based technique: as the

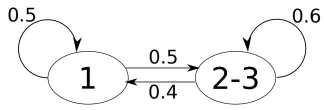

Figure 7: Revised version of Figure 6, compressed by collapsing states 2 and 3 to a single state.

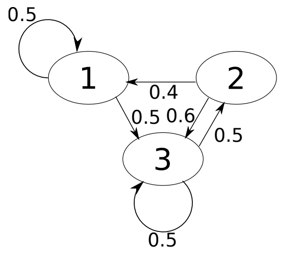

Figure 8: Revised version of Figure 6, compressed by removing transitions with low probability.

model becomes simpler, each portion of the model becomes supported by more data. Therefore, more evidence supports each point of the model, increasing the statistical soundness of the model and minimising errors due to extrapolation.

Practically speaking, compressing the model such that the evidence supports the model is accomplished by modifying the question that is being asked of the model. Hence, instead of attempting to find an exact model of the system, for which there is insufficient evidence and therefore there is little confidence in the correctness of the model, an approximation is found. As the approximation has less precision than the actual model, there is less of a burden on evidence gathering and hence less evidence is required to support the model. To illustrate this in terms of the WCET problem, the uncompressed model simply lists all available paths through the program, and finding the worst case path with any confidence requires measuring each path. The model can be compressed by identifying features of these paths to establish groups; these groups can then be sampled, and hence evidence gathered on which group of paths contains the worst case path. As evidence is gathered per group and not per path, less evidence is required to produce a statistically sound model.

As previously stated, the ideal Markov Chain model consists of states corresponding to a number of consecutive budget overruns, with a probability table dictating the probability of the next state. This simple description gives two forms of information to work with:

- *States and Annotations*: The states annotation, α_s, gives the number of consecutive budget overruns. By lowering the precision of α_s from a specific duration of budget overruns, to a bucket of durations for budget overruns, as well as merging the appropriate Markov States, the model becomes simpler and requires less information to populate it. However, such a change comes with a corresponding loss of accuracy as the exact number of consecutive budget overruns will no longer be known. An example of this is given in Figure 7.
- *Probability Table*: The probability table within each state, denoted by the rows $tr \in states[\alpha_s]$ must be constructed based on evidence provided. If there is insufficient evidence to support a state transition probability in the probability table, then this state transition can be merged with a sufficiently similar (although pessimistic) state transition to remove the offending evidence, albeit with a loss of accuracy. An example of this is given in Figure 8.

In order to compress the ideal Markov Chain model into a more manageable and smaller model, states and annotations are compressed such that instead of an exact value, the state and annotation correspond to a bucket containing a range of consecutive budget overruns. When fitting the Markov Chain to data, an event leading to n consecutive budget overruns is categorised in the bucket which contains n. While this obviously represents a large drop in accuracy, it also lowers the amount of test data needed for learning considerably. However, other than the adaptation to buckets, the input test data is used to inform the probability tables just as in the ideal case. Provided that the user can provide a series of specific numbers of consecutive budget overruns which are interesting for the system, buckets can be constructed to provide the maximum possible accuracy for these values. For example, if a user is interested in events causing at least 10, 20, or 30 budget overruns, appropriate buckets are $[1, 9]$, $[10, 19]$, $[20, 29]$, and $[30, maximum - overruns)$. In general, if a user is interested in events causing at least $a_1, a_2...$ budget overruns, the appropriate buckets are given by partitioning the range $[0, maximum - overruns]$ by the values $a_1, a_2....$ Of note is that the buckets are the only input that a user must provide to the compression; all other compression employed is fully automated.

The next stage in compressing the Markov Chain is to compress the probability table. It is expected that the amount of data points contributing to each entry in the probabilities tables will vary significantly. As some entries may not have sufficient data points to have confidence in their results, action must be taken. This results in modification of the probability table such that values which do not have sufficient data to have confidence are merged into an appropriate and pessimistic set of values. For example, if there are only a few observed values for the transition to state α_1, then the observed values are merged into the transition to state α_2, where the state α_2 is more pessimistic than α_1. This is repeated as necessary, unless α_1 is the state corresponding to the largest bucket; in this case compression cannot continue and the user would have to provide additional data as insufficient data is present for the model to have the desire statistical confidence. As the compression always increases the expected duration of a fault, the compression cannot introduce unsound results. When this compression is successful, as the user has provided information on the intended use case, the loss of precision is expected to have minimal effect on the usefulness of the model. This can be tested by examining the accuracy of the extrapolation with and without compression; when compression is enabled, it is expected that the results will be more accurate when compared to real data, due to the questions being answered becoming simpler. However, one should take care that the model is not overly compressed, as this will result in the questions the model can answer being so simple that they provide no useful information. In practice, the degree of compression can be automatically determined by comparison of the desired structure with the structure used in the model, and in the case of overcompression the user can be alerted.

More concretely, this process is defined in Algorithm 1, *LearnLC*. At this level, the fault durations have already been extracted from observed data, and so the only task is

```
1  $minSamp \leftarrow$ minimum sample size needed for statistical significance
2  Function LearnLC(buckets, faultDurations)
3      create an empty Markov Chain model states using the labels $\alpha_s$ from buckets
4      $compressed \leftarrow set()$
5      $curentState \leftarrow$ the label $\alpha_0$ of the initial state of states
6      for $fd$ in $faultDurations$ do
7          $states[currentState][bucket(fd)] += 1$
8          $currentState \leftarrow bucket(fd)$
9      for $\alpha_s$ in states do
10         for $tr$ in $states[\alpha_s]$ do
11             if $states[\alpha_s][tr] < minSamp$ then
12                 if $tr \neq max(states[\alpha_s])$ then
13                     $tr' \leftarrow$ least value in $states[\alpha_s] > tr$
14                     $states[\alpha_s][tr'] += states[\alpha_s][tr]$
15                     $states[\alpha_s][tr] \leftarrow 0$
16                     add $(\alpha_s, tr, tr')$ to compressed
17                 else
18                     return None, Data is unusable
19     for $\alpha_s$ in states do
20         $total \leftarrow sum(states[\alpha_s][tr]$ for $tr$ in $states[\alpha_s])$ for $tr$ in $states[\alpha_s]$ do
21             $states[\alpha_s][tr] \leftarrow states[\alpha_s][tr]/total$
22     return $states, compressed$
```

Algorithm 1: The *LearnLC* function, which uses Lossy Compression to fit a Markov Chain to represent input data

to construct the Markov Chain model. Line 6 takes the observed fault durations and works out which transitions are taken in the Markov model; this data is then stored in appropriate counters. Line 9 takes this Markov model and applies appropriate compression to any transition between states which fails to have enough samples for the desired statistical confidence. This algorithm applies compression in a pessimistic manner, by assuming that a greater number of faults will occur (Line 12); if such a transition cannot be found, then the compression fails and as the model can no longer be constructed with the desired confidence, the model is considered unusable (Line 18). Finally, Line 19 takes the observed tallies of transitions and converts them to the probability tables required for a Markov Chain. These probability tables are then returned, along with a description of which states were compressed. An illustration of a sample fault model produced by this method is given in Figure 9.

Summarising, this section gives the *LearnLC* method to fit a Markov Chain model to a set of budget overrun durations, which given the limited variability between consecutive job execution times, is expected to provide an adequate model of budget overruns. However, test data is still unlikely to contain sufficient budget overruns to provide enough data to fit the model. Hence this method must be applied to data encountered with an artificially lowered budget overrun threshold, for which there will be much higher amounts of data, and these results used to infer the model for actual budget overruns. An overview of how this is accomplished is given in the next section.

6. FAULT THRESHOLDS AND BUDGET OVERRUNS

Given the lack of data available for the system when it has failed, it is necessary to determine a method to infer this data. As previously stated, due to the assumption that when deployed, inputs between consecutive jobs have limited variability, and therefore consecutive job execution times have limited variability. This results in the kind of distribution seen in Figure 5, where budget overruns occur

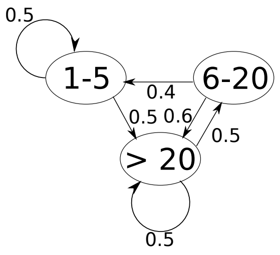

Figure 9: A Sample Fault Model

at the peaks of computationally intensive periods.

In order to prepare the raw data for use in creating forecast models, it is necessary to provide multiple points of observation. This is accomplished by setting multiple thresholds for computationally intensive jobs, as illustrated in Figure 5, which unlike a WCET budget exceedance, provide sufficient amounts of observations. However, due to the compression applied by the *LearnLC* method, it is possible that Markov Chains with different structures will be produced. As this complicates the forecasting process, the simplest method to counter this is to produce as many models as possible, and then select the most commonly occurring model structure. This process is given in the *GatherData* function, given in Algorithm 2. Specifically, *GatherData* iterates over every possible threshold level and uses a map (Line 2) to keep track of what model structures are seen (Line 6).

```
1  Function GatherData(buckets, rawData)
2      $learntData \leftarrow map()$
3      for $x$ in $0...max(rawData)$ do
4          $durations \leftarrow$ list of contiguous elements of $rawData$ all greater than $x$
5          if LearnLC(buckets, durations) is not None then
6              $states, compressed =$ LearnLC(buckets, durations) if compressed not in learntData then
7                  $learntData[compressed] = list()$
8              append $(x,$ states) to $learntData[compressed]$
9      return longest list in values of learntData
```

Algorithm 2: The *GatherData* function, which gathers observations from execution time data

After the data for each point is found, forecasting is applied by fitting an appropriate curve to the data. This is accomplished by using the Levenberg-Marquardt algorithm [21], otherwise known as the Damped Least-Squares method, to fit a variety of polynomial curves and choosing the one with the lowest order which gives a sufficiently accurate fit. This selection criteria is due to the fact that higher order polynomials are subject to concerns about overfitting the data, which causes extrapolation to give wildly inaccurate results.

It can reasonably be hypothesised that the parameters of the Markov Model converge; this is due to the fact that as thresholds increase, the durations of faults decrease. This in turn leads to a smaller number of paths in the Markov model being taken, and hence the parameters will converge. In turn, this means that it is expected that non-linear curves which converge on a fixed value (e.g. hyperbolic curves) are an appropriate model. This also has the advantage that as the curves converge on a fixed value, errors from extrapolation are minimised. For completeness, other types of curve are considered by the algorithm. However, one consideration of fitting a non-convergent curve is that extrapolation becomes much more prone to errors, and hence the correctness of the input data becomes

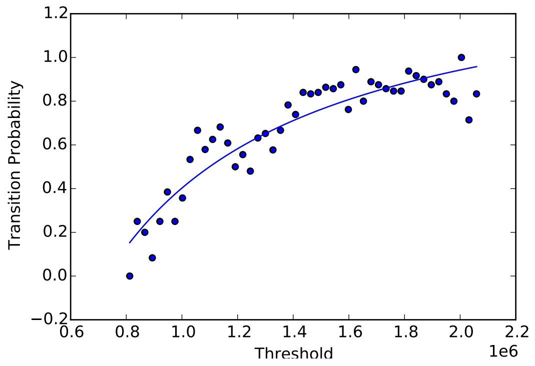

Figure 10: Fitting Curves to Probability of Transition from State 15-25 to State 5-10

much more important. Finishing the formal definition, the *Extrapolate* function, shown in Algorithm 3, generates the extrapolated Markov Model.

```
1 Function Extrapolate(buckets, rawData, threshold)
2     create an empty Markov Chain extrapolatedStates
3     for states in GatherData(buckets, rawData) do
4         for each Markov Chain parameter in states, x do
5             Fit a curve to the plot threshold, x
6             Set the corresponding Markov Chain parameter
              in extrapolatedStates to the extrapolated value
              at threshold
7     return extrapolatedStates
```

Algorithm 3: The *Extrapolate* function, which generates an Extrapolated Markov Model

7. EVALUATION OF FORECAST MODELS

As previously mentioned in Section 3, data for experimentation was extracted from FFmpeg [7], using Callgrind [8] to extract the number of instructions taken to decode P-frames from a video. As in previous examples, the video used for this evaluation is *Return of the Kung Fu Dragon*, an input which has no special properties other than being a valid and typical input to FFmpeg. This is in contrast to other sources, for example encoded random noise, which are not representative of the types of input that FFmpeg would be expected to handle in normal operation. In order to evaluate the effectiveness of the forecasting techniques employed, modelling is conducted using a fraction of the available data. This allows the extrapolation to be compared against the complete data set, and its accuracy to be evaluated.

In this experiment, 5%, 7% and 10% of the data of the film was used to learn a Markov model using the rules given in Sections 5-6. These amounts of data were chosen as they provide a statistically significant amount of data to the learning process, but also reduce the amount of data required for learning significantly.

In this experiment, fitting curves is relatively straightforward; all curves are similar to that seen in Figure 10, and admit a hyperbolic curve with minimal error. Excluding the initial data (below 110000 instructions), the data points are displaced from the curve by an independent random amount, which is indicative of random sample variation (as opposed to data which is displaced from the curve by a non-random or dependent variable, which is indicative of systematic sampling errors). While there may be cases where a different curve is needed to provide an accurate forecast, this demonstrates that curve fitting is an appropriate forecasting method in this case.

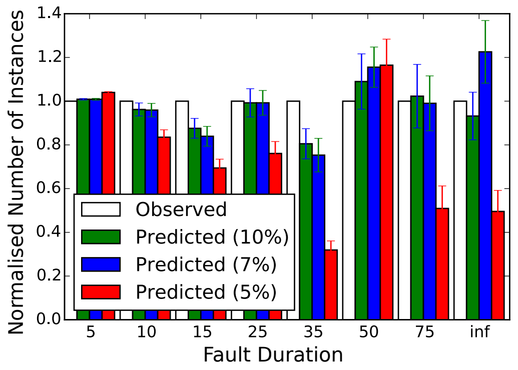

Figure 11: Forecast and Actual Consecutive Budget Overruns at threshold 1600000 for Return of the Kung Fu Dragon

As expected, longer fault durations produce less data. Hence, lossy compression identified that data on longer fault durations was scarce and reduced the number of buckets used for the longer durations. This gives the desired effect of trading accuracy of the model for statistical confidence; by contrast, if the model was not compressed the buckets for longer durations would contain less than 15 data points each, which cannot be argued as being statistically significant. Hence the *LearnLC* method was capable of finding a model for which there is sufficient data to give statistical significance to each aspect of the model.

In addition, to verify the expected result that lossy compression improves the confidence of the model's forecast by simplification of the questions the model seeks to answer, the previous experiment was repeated with lossy compression of the probability tables disabled. In this case the errors produced by the model were incredibly high, due to some entries in the probability tables having very little evidence (less than 5 samples), and hence confidence in such a model is low.

Examining the specific results of the model, Figure 11 compares the results of the extrapolated Markov model, when trained with 10%, 7% and 5% of the data respectively, for forecasting the frequency and duration of budget overruns, defined to be at 1600000 cycles. The graph illustrates the mean and standard deviation of model when compared to the observed data. As expected, lowering the amount of data used to train the model results in a decrease in accuracy. This can be seen in the results as while the error on the model trained with 10% of the data is normally in the region of 10%, as the amount of data decreases the error increases. However, in most cases the model is not wildly divergent from the actual behaviour, and thus provides a useful fault model. This demonstrates that the techniques used to construct the model, lossy compression, Markov Chains and extrapolation, are capable of producing a useful description of the behaviour of faults even if there is little data available about these faults.

There is however one caveat to these results, which is that the data includes 2 computationally intensive segments, caused by fading scene transitions in the input video, which are not representative of normal operation. The primary reason for the divergence of the 5% trained model is that unlike the 7% and 10% models, it only

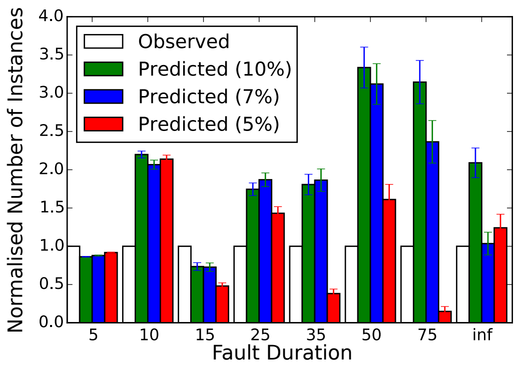

Figure 12: Forecast and Actual Consecutive Budget Overruns at threshold 1600000 for Return of the Kung Fu Dragon, excluding 2 computationally intensive areas

contains a portion of one of these computationally expensive segments. Figure 12 shows the results when the 2 computationally expensive segments are excluded (analogously to the exclusion of startup times in other forms of analysis). In this case, the model becomes substantially more pessimistic, typically forecasting longer faults than the observed data; this is explained by the fact that as the computationally expensive portion has been removed, the number of elements being placed in longer duration bins has decreased and therefore more compression has taken place (to gain a significant sample), resulting in higher pessimism. However, in this case the data which has been used to train the 5% model is now more representative of the actual data, and therefore the model no longer diverges as much from the other forecast models. This leads to a conclusion that the lower the variability of the data, the less data is needed to get a consistent result; hence by examining the variability of the input data, an appropriate sample size is then found. However, it also highlights that the buckets used must be chosen carefully; while in this case the buckets have been kept consistent to allow direct comparison of the results, as the buckets are selected inappropriately for the data, the additional compression results in a more pessimistic result.

8. DEPET: EXECUTION TIMES WITH DEPENDENCIES

Research has been conducted into effectively splitting a given processor utilisation into uniformly distributed task utilisations, for example the UUniFast algorithm [3], and extensions to multicore UUniFast-discard [10] and RandFixedSum [11]). Such algorithms are designed to be used in the testing and development of scheduling algorithms. In the case of UUniFast, given that scheduling algorithms are expected to handle a wide variety of tasksets, the use of a uniformly distributed set of test data is appropriate to find any corner cases.

While these methods, combined with approaches for task period generation, are appropriate for determining a set of WCETs, they do not provide a means of generating the execution times of jobs at runtime. Indeed, there has been little research on a model for the temporal dependencies of job execution times within a system, with the most advanced approaches [6] only allowing very specific forms of dependencies. However, evaluating the performance of scheduling algorithms on realistic scenarios (i.e. finding the number of times a processor enters its low power state due to tasks not reaching their WCET), or evaluating the performance of the mode change algorithm in a mixed criticality system [4, 1] would benefit from such a generator.

Utilising the previous results on forecasting exceedance durations, it is possible to construct a model for execution times which includes dependencies between execution times of runs of a *task*, which can be used to generate job execution times in a task simulator (Figure 1). The new approach, **DepET**, uses multiple forecast models to determine how the model generates execution times, while maintaining an appropriate overall distribution.

The use of multiple forecast models is accomplished by splitting the execution times of the task into a series of *bands*, which allow distinct behaviours of the task to be identified and expressed independently. When generating an execution time for a given task, that task's currently selected band is used to determine the parameters of the execution time. In addition, when sampled each band has a probability of causing the execution times of the task to move into the next band, for a duration determined by an exceedance model, such as the exceedance models given in the previous section.

- mn, mx: The minimum and maximum values within this band
- d: The maximum value that an execution time may be displaced from it's previous instance
- p: The probability of leaving the band
- EM: An exceedance model, used to determine the duration of entering a higher band
- $prev, next$: Pointers to the previous and next band.
- $duration$: A counter to store the number of samples to take from this or a higher band; when the algorithm runs, the duration of each band is decremented on each generated value
- cET: Holds the current execution time value

While the p value of a band can be set by the user (for example, if the user has knowledge on the system), in order to match an overall distribution defined by a probability density function P bounded above by d_{mx}, an approximate[2] p value for each band is as follows:

$$band.p = \left(\frac{P[task.mn \leq X \leq task.mx]}{P[task.mx \leq X \leq d_{mx}]}\right)^{\frac{2}{no\_quantiles}} \quad (1)$$

Tasks are primarily characterised by the bands which comprise them. In the *DepET* algorithm, these bands are given produced based on the statistical distribution of the training data, with the n bands being picked to denote the n-quantiles[3] of the training data. For each of these bands, the exceedance model of the band is defined to be the exceedance model predicted by the extrapolation prediction method for the maximum value of the quantile. In addition, to account for data which is predicted but not seen in the training data, additional bands can be added; it should be noted that due to the nature of the predicted models, the further these bands extend from the training population, the less accurate the predicted exceedance model will be. In addition, as mentioned in the overview of

[2]This approximation was found by search, but is much simpler than the computation for an accurate value.

[3]n-quantiles: the set of values which divide the population into n components of equal size.

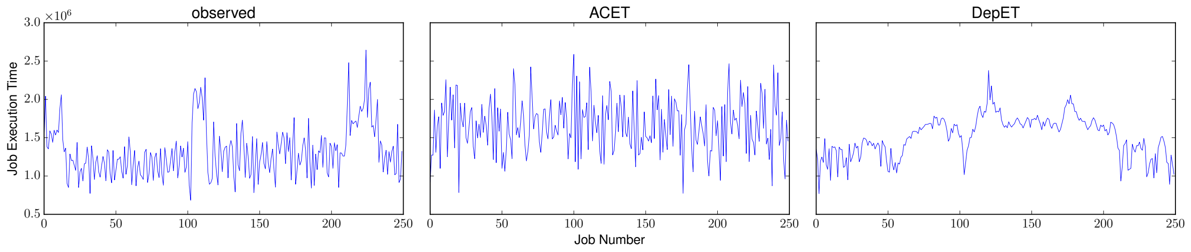

Figure 13: Temporal Distribution of Execution Times

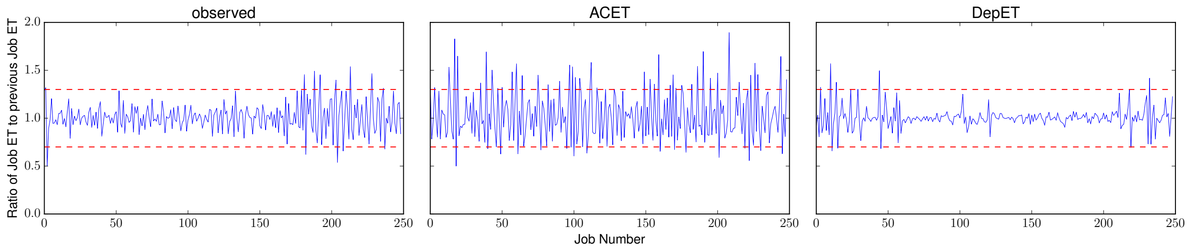

Figure 14: Ratio of Job Execution Time with Previous Instance

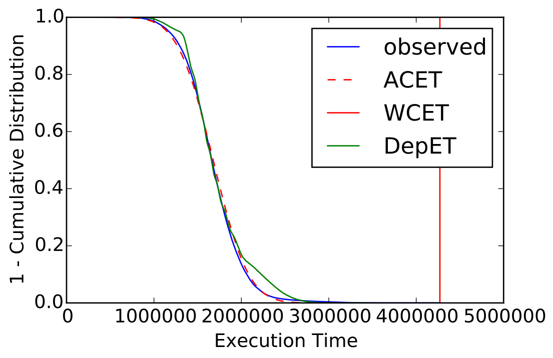

Figure 15: Overall Distribution of Execution Times

```
Function DepET(tasks)
    $ETs \leftarrow []$
    for $task \in tasks$ do
        $band \leftarrow task.current$
        add $randomnormal() * band.d$ to $band.cET$
        $ETs.append(band.cET)$
        clamp $band.cET$ within $band.mn$ and $band.mx$
        if $band.duration = 0$ then
            $task.current \leftarrow band.prev$
        else if $random() < band.p$ then
            $task.current \leftarrow band.next$
            $band.next.duration \leftarrow band.EM.sample()$
        while band is not None do
            decrement $band.duration$
            $band \leftarrow band.prev$
    return $ETs$
```

Algorithm 4: The *DepET* algorithm

the algorithm, tasks also contain state to indicate the current band that the task is utilising for execution time generation. Hence the tasks are defined with the following properties:

- *bands*: The bands which generate execution times for the task
- *current*: The current band being used for generation of execution times

Having given the overall operation of the algorithm, Algorithm 4 gives a concrete definition.

9. EVALUATION OF DEPET

In order to demonstrate the properties of DepET, a simple evaluation was carried out against the ACET method implemented in SimSo [6]. This method was picked as this analysis focuses on a single task and most other methods in SimSo focus on the effects of pre-emption and task migration, which clearly do not have an impact in the single task case. To carry out the evaluation, the same data set was used as in the previous training example. In order to instantiate the variables in the DepET and ACET methods, appropriate measurements were taken so that these methods approximate the input data.

As seen in Figure 15, all methods are capable of producing an overall distribution similar to the input distribution. There are minor discrepancies with the ACET method due to the assumption that average execution times are normally distributed. On the other hand, DepET is capable of producing a very similar distribution, although with errors around band transitions.

The advantage of DepET over the ACET methods becomes clear when examining the temporal distribution, shown in Figure 13. In the temporal distribution, the ACET methods only shares the distribution of the magnitude of execution times with the training data; temporal dependencies are not expressed. However, temporal dependencies can be clearly seen in DepET, and this is highlighted in Figure 14, which shows the ratio of a (simulated) jobs execution time to the previous instance. Although due to the effects of bands DepET struggles to capture some of the variability when the chance of band transition is high, the characteristics of DepET are much more similar to the observed data. In particular, the number of times that the ratio exceeds 30%, marked on the graphs, is far higher in the ACET method than with DepET or observed data, indicating that DepET provides a much more realistic model of dependencies.

10. CONCLUSIONS

This paper presented a method to model dependencies in job budget overruns by utilising Markov Chains, Lossy Compression and Forecasting methods to create a model of budget overrun durations. Markov Chains were demonstrated to provide a suitable representation of the system, Lossy Compression was used to ensure that all observations were of statistical significance, and Forecasting used to combat the rarity of observations of budget overruns in testing. The resulting algorithm was tested using a controlled experiment, and provided that sufficient data was available, produced a forecast accurate to within 10% of the actual observed data.

In addition, a new algorithm for the generation of job execution times, *DepET*, was defined. *DepET* addresses the need for a realistic execution time generator for use in task simulators, by combining multiple exceedance models to control the generated execution times. This approach delivers clear dependencies between instances of tasks, while respecting the overall profile of the task. As DepET is expected to be employed after being trained on a relatively small set of data, the effect of DepET is to enable a task simulator to effectively mimic the properties of an actual task, rather than relying on a synthetic random distribution of execution times which is unrealistic due to an assumption of independence.

Further work is needed to verify the effectiveness of this approach with other sources of data, in particular when the input data has a higher degree of variability. In addition, the forecast modelling methods presented in this paper are currently being adapted to take into account additional sources of information, such that a model of execution times can be derived for a given circumstance. This is expected to be useful in cases such as incremental development, as the model could be used to determine the impact of changes to one task on other tasks with minimal retesting. This is especially useful given the challenges of development on multicore platforms.

Acknowledgement

This work was partially funded by the EU FP7 Integrated Project PROXIMA (611085) and EPSRC MCC Grant EP/K011626/1 and the Inria International Chair program. EPSRC Research Data Management: No new primary data was created during this study. We acknowledge the Swedish Foundation for Strategic Research (SSF) SYNOPSIS Project for supporting this work.

11. REFERENCES

[1] I. Bate, A. Burns, and R. I. Davis. A bailout protocol for mixed criticality systems. In *27th Euromicro Conference on Real-Time Systems*, 2015.

[2] G. Bernat, A. Burns, and A. Llamosi. Weakly hard real-time systems. *IEEE Trans. Comput.*, 50(4):308–321, April 2001.

[3] E. Bini and G. C. Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems*, 30(1-2):129–154, 2005.

[4] A. Burns and R. Davis. Mixed criticality systems-a review. *Department of Computer Science, University of York, Tech. Rep*, 2013.

[5] A. Burns and A. Wellings. *Real-Time Systems and Programming Languages: Ada, Real-Time Java and C/Real-Time POSIX*. Addison Wesley, fourth edition, May 2009.

[6] M. Chéramy, P. E. Hladik, A. M. Déplanche, and S. Dubé. Simulation of real-time scheduling with various execution time models. In *Proceedings of the 9th IEEE International Symposium on Industrial Embedded Systems (SIES), Work-in-Progress session*, 2014.

[7] Contributors. FFmpeg. https://www.ffmpeg.org/.

[8] Contributors. Valgrind tool suite. http://valgrind.org/.

[9] L. Cucu-Grosjean, L. Santinelli, M. Houston, C. Lo, T. Vardanega, L. Kosmidis, J. Abella, E. Mezzetti, E. Quiñones, and F. J. Cazorla. Measurement-based probabilistic timing analysis for multi-path programs. In *24th Euromicro Conference on Real-Time Systems*, pages 91–101, 2012.

[10] R. I. Davis and A. Burns. Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. *Real-Time Systems*, 47(1):1–40, 2011.

[11] P. Emberson, R. Stafford, and R. I Davis. Techniques for the synthesis of multiprocessor tasksets. *Proceedings 1st International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS 2010)*, pages 6–11, 2010.

[12] N. Fenton, M. Neil, and D. Marquez. Using bayesian networks to predict software defects and reliability. In *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability*, volume 222, pages 701–712. SAGE Publications, 2008.

[13] R. Fildes. The evaluation of extrapolative forecasting methods. *International Journal of Forecasting*, 8(1):81 – 98, 1992.

[14] P. Graydon and I. Bate. Safety assurance driven problem formulation for mixed-criticality scheduling. In *Proceedings of the 1st International Workshop on Mixed Criticality Systems*, pages 19–24, 2013.

[15] P. Graydon and I. Bate. Realistic safety cases for the timing of systems. *The Computer Journal*, 57(5):759–774, 2014.

[16] D. Griffin. *Lossy Compression applied to the Worst Case Execution Time Problem*. PhD thesis, University of York, 2013.

[17] D. Griffin, B. Lesage, A. Burns, and R. I. Davis. Lossy compression for worst-case execution time analysis of plru caches. In *Proceedings of the 22nd International Conference on Real-Time Networks and Systems*, RTNS '14, pages 203–212, 2014.

[18] D. Griffin, B. Lesage, A. Burns, and R. I. Davis. Static probabilistic timing analysis of random replacement caches using lossy compression. In *Proceedings of the 22Nd International Conference on Real-Time Networks and Systems*, RTNS '14, pages 289–298, 2014.

[19] Z. A. H. Hammadeh, S. Quinton, and R. Ernst. Extending typical worst-case analysis using response-time dependencies to bound deadline misses. In *Proceedings of the 14th International Conference on Embedded Software*, EMSOFT '14, pages 10:1–10:10, New York, NY, USA, 2014. ACM.

[20] W. K. Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

[21] J. J. Moré. The Levenberg-Marquardt algorithm: implementation and theory. In *Numerical analysis*, pages 105–116. Springer, 1978.

[22] I. Wenzel, R. Kirner, B. Rieder, and P. P. Puschner. Measurement-based timing analysis. In *ISoLA*, pages 430–444, 2008.