Using Digital Twins in the Development of Complex Dependable Real-Time Embedded Systems

Xiaotian Dai, Shuai Zhao, Benjamin Lesage, Iain Bate

Department of Computer Science, University of York, UK
{xiaotian.dai, shuai.zhao, benjamin.lesage, iain.bate}@york.ac.uk

Abstract. Modelling execution times in complex real-time embedded systems is vital for understanding and predicting tasks' temporal behaviour, and to improve the system scheduling performance. Previous research mainly relied on worst-case execution time estimations based on formal static analyses that are often pessimistic. The models that resulted are hard to maintain and even harder to validate. In this work, the novel use of Digital Twins provides opportunities to improve this issue and beyond for dependable real-time systems. We aim to establish and contribute to three questions: (i) how to easily model execution times with an adequate level of abstraction, and how to evaluate the quality of that model; (ii) how to identify errors in the models and how to evaluate the impact of errors; and (iii) how to make decisions as to when and how to improve the models. In this paper, we proposed a Digital Twin-based adaptation framework, and demonstrated its use for modelling and refining execution time profiles. Key decisions concerning the quality of the model and its impact on performance are evaluated. Finally, some challenges and key research questions for the formal method community are proposed.

Keywords: Digital Twin · Real-Time Embedded Systems · Execution Time Model · Error Modelling · Error Refinement · System Adaptation.

1 Introduction

For complex real-time embedded systems (RTES), modelling of the execution times is essential as it helps to understand and predict a system's temporal behaviour, validating the timing requirements, and to improve the system performance. In RTES, the execution times can be largely affected by a number of factors, to name a few: the program execution path, which is based on the current inputs; the interference and blocking from other sources, *e.g.,* co-running tasks, and contentions due to accessing shared resources; the underlying hardware and architectural features on which the program is executed; and the current system mode, and in some contexts is subjected to operational scenarios, *etc.*

In the real-time systems community, the widely applied practice is to derive the worst-case execution time (WCET) of a program to understand its worst-case performance, with static analyses applied based on the control flow graph and the hardware and memory model [19]. However, this formal approach is pessimistic and lacks the ability to adapt to changes and to mitigate faults when there exists model inaccuracy due to partial information or change of system, *etc.* The models that resulted are hard to maintain and even harder to validate. As the WCET fits in with a larger, more complex problems including worst-case response time and task scheduling and allocation, consequently the methods based on the WCET model can be both too pessimistic that produce low resource utilisation, and being fragile to uncertainties or violation of any assumptions that are used to derive the model.

In recent years, there is an increasing trend to apply Digital Twins (DT) in automotive, aerospace, manufacturing, transportation and healthcare systems. The idea of a DT is to establish a digital representative of the target system that is largely based on models and simulations. We recognise the potential of DTs in the design and development of real-time embedded systems.

In this paper, we apply Digital Twin as a first step towards improving the adaptiveness, accuracy and dependability in RTES. We are interested in applying it to both critical systems, *e.g.,* avionics (HiClass[1]), and more conventional systems, *e.g.,* telecoms (MOCHA[2]). Our DT work continues the success of model-based design for embedded systems where previously it was largely based on off-line simulations and verification, while lacking the ability of on-line adaptation. In our case, the Digital Twin is running in parallel with the physical entity at the same time while the target system is in operation. Data carrying the information of the target system and decision from the Digital Twin exchanges between the physical system and the digital counterpart in real-time. The novel use of Digital Twin in this work provides opportunities to be adapted to multi-core real-time systems from the following aspects:

– to correct timing models in which inaccuracies exist that could reduce performance or invalidate dependability;
– to suggest improvements to scheduling policies based on evidence obtained through observation;
– to make on-line decisions relevant to scheduling (*e.g.,* admit new tasks to schedule or allow more events to be processed);
– to support off-line assurance decisions through large-scale simulations.

Without losing generality, we focus on the timing aspects as it is recognised as the most critical factor in RTES. Specifically, we started from an execution time model based on task-level cache reuse, the cache recency profile (CRP), as an alternative to the WCET. The CRP describes the benefit in terms of execution time speedup a task can have based on a warmed cache from previous executed job instances. It is a sensitivity model of execution times in terms of how many cache lines have been accessed since the task of interest was last executed.

However, like other models, the CRP is sensitive to the current workload and operational context. Thus, it can be inaccurate if the context is shifted from the context in which the model was originally produced. The task scheduling and allocation algorithm is then based on this CRP. Associated with the CRP is an error model which represents the differences between the real system and the simulated part of the Digital Twin. The error model indicates the misalignment of models and supports analysis to understand robustness or resilience of the task scheduling and allocation of the system.

In this paper two key challenges are exploited: what should the components and models in the Digital Twin feature (*e.g.,* level of abstraction and key features

[1] `https://www.cs.york.ac.uk/news-events/news/2020/hi-class/`
[2] `https://www.cs.york.ac.uk/rts/mocha/`

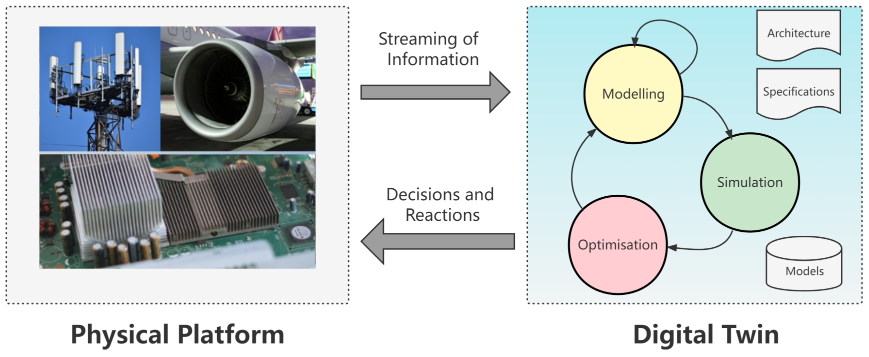

Fig. 1. The DTiL-RTES framework for complex real-time embedded systems

that exist) and how the key models (*e.g.,* the parameters associated with the model and the model itself) are refined based on decisions made with them. In this paper, this modelling paradigm with Digital Twin and key decisions concerning the modelling accuracy and its impact are evaluated.

Contributions: In this work, we formulate a model-based Digital Twin framework for multi-core embedded systems to improve the system performance and resilience. We aim to establish and contribute to the following key research questions (RQs):

– RQ. 1. How to easily model execution times with an adequate level of abstraction, and how to evaluate the quality of that model against observations.
– RQ. 2. How to identify errors in the models and how to evaluate the impact of errors on key performance indices.
– RQ. 3. How to make decisions as to when and how to improve the models, supported by evidence collected from the real system and/or a simulator.

We propose the concept of *Digital Twin in-the-Loop design for real-time embedded systems* (DTiL-RTES), as shown in Fig. 1. The idea is to extend the use of Digital Twin for real-time and embedded systems beyond the design process, and exploit its usage at operational time by formulating an observation-decision loop. The DTiL-RTES performs both static and dynamic profiling, while making predictions from simulation and based on models of the system. Although there are many potentials of applying Digital Twin in multi-core systems, we focus on timing perspective for scheduling and allocation particularly for this paper. The DTiL-RTES approach is designed to mitigate model inaccuracy given extra information and/or derivation is observed while the system is in operation. It improves modelling accuracy by examining the results of the actual system with predictions based on the models. The DTiL-RTES can also be easily extended with a run-time monitor and an anomaly detector.

Organisation: the proposed DTiL-RTES framework is introduced in Sec. 2, which is followed by modelling and refining execution time models in Sec. 3. The evaluation is then given in Sec. 4, with discussions on limitations and challenges for the formal methods community. The related work on execution time modelling is introduced in Sec. 5, with concluding remarks in Sec. 6.

2 The DTiL-RTES Framework

In this section, we introduce the proposed DTiL-RTES framework, including its design intuition and components. We will then give more details in Sec. 3 on the execution time model and error refinement.

2.1 Design Philosophy

A Digital Twin is defined as a digital replica of a target physical system (*i.e.,* the system of interest) [5]. It can run independently and/or in parallel with the target system to facilitate making predictions and/or decisions. From a modelling perspective, the Digital Twin combines models, and methods based on the models to simulate, predict, analyse, and evaluate. The Digital Twin is normally running on a more capable machine (*i.e.,* the host) other than the target system. It is normally built based on existing services that are already established between the system and the host.

Efforts of using a DT in real-time systems for run-time WCET modelling and parameter adaptation were early discussed in [3,4]. The requirements and open challenges of adapting DT to the context of multi-core real-time systems have been discussed in [5]. Different to a cycle-accurate simulator, a Digital Twin often focuses on a higher level of abstraction where the key characteristics are identified. In general, the proposed DT is designed to meet the following required purposes: (1) answering what-ifs (decision making and optimisation) [7]; (2) support continuous modelling [3]; (3) understanding outliers and detecting anomalies. In addition, abnormal behaviours can be observed, which provides useful information to, for example, studying online and offline scheduling and allocation policy in embedded real-time systems. It is recognised that these concepts in Digital Twin provide a foundation and form a general background of this work, as well as the design and modelling philosophy.

2.2 DTiL-RTES Overview and Components

An overview of the DTiL-RTES framework proposed in this work is shown in Fig. 2. Key functions of the DT are: (a) decision support at both design-time and run-time; (b) anomaly detection to indicate inadequacy of the model; (c) tuning models based on new/novel observations. The DTiL-RTES consists a number of components. We introduce the key components of the proposed DT as follows.

Specification, System and Task Models: the specification and models are provided as database files to be used by the DT. Specifications include: (i) system-level requirements, *e.g.,* deadline miss rate requirement; and (ii) task-level requirements, *e.g.,* response time and jitter requirements. The models include: (i) system model, *i.e.,* processor and memory model, including processor grouping, frequencies, memory hierarchy, *etc*; (ii) task model, provide properties of the task set, including inter-task dependencies, period, deadline and the worst-case execution time of the tasks; (iii) Execution Time Models: models to predict the execution times of tasks given system input states. There is also an associated error model to include factors that are not included in the execution time model.

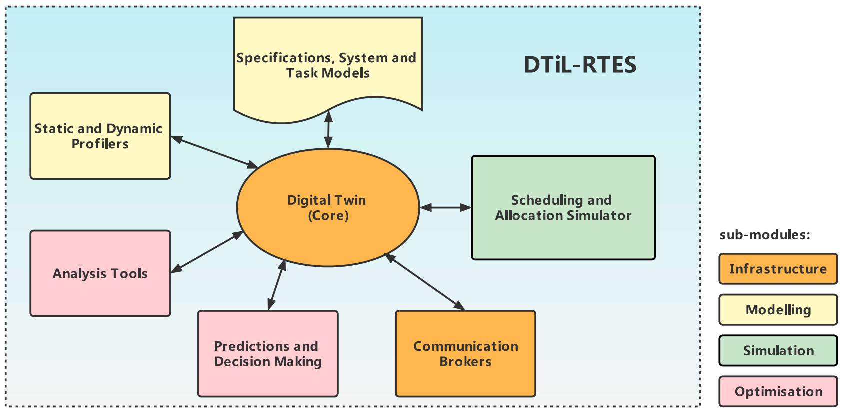
DTiL-RTES
Specifications, System and Task Models
Static and Dynamic Profilers
Digital Twin (Core)
Scheduling and Allocation Simulator
Analysis Tools
Predictions and Decision Making
Communication Brokers
sub-modules:
Infrastructure
Modelling
Simulation
Optimisation

Fig. 2. An overview of the Digital Twin in-the-loop design concept

Scheduling and Allocation Simulator: simulates the behaviour of the scheduler, by interpreting the processor, task and execution time models into a high-level simulator. The key is to understand what is the impact that a change in the model has on the scheduling decisions and performance.

Analysis Tools: accesses data from the system, streaming data to a local or remote database, and then applies statistical analysis upon the observations. For strictly hard real-time tasks, a formal schedulability test has to be checked against deadline constraints, for example, using response time analysis. As part of the DTiL-RTES, if it is established that a change is needed to a task's execution time model then the impact of this change is also checked.

Predictions and Decision Making: the decision making process applies changes to the model based on results from the scheduling simulator and, optionally, schedulability analysis. The decision process can either be offline or online. Examples of decisions include make changes to models, use a more advanced classifier, track more objects and accepts more incoming data streams, *etc*.

Communication Brokers: the module to connect the Digital Twin host with the Digital Twin client(s). The module maintains communication for data passing between the host and the client(s), and with defined Quality-of-Service (QoS), *e.g.,* bandwidth limitation.

Note that the complete DTiL-RTES framework has other components for other (non-timing-related) purposes, for example, safety argument, diagnosis and fault identification. Due to the limitation of pages and scope, we will not give an exclusive list but ignore modules irrelevant to this work.

2.3 Intended Use and Overhead

The position of the DT in this work focuses on building and refining a predictive timing model, named as Execution Time Model (ETM). To be more specific, an ETM is used to produce predictions of the execution time of a program, given inputs including the system states, data inputs, system modes, *etc*.

The ETM advances the WCET model in the way that it produce run-time predictions in addition to the worst-case single estimation of the execution time, thus the scheduler can make better use of the CPU resources without being too pessimistic from the overly assumed worst-case scenario. The ETM model can be built offline with data from running the system in a test environment. However, it is notable that there are many factors that will make the original ETM (*i.e.,* the model built offline) imprecise. Examples of these factors include insufficient information of the system, dynamically changing environment, contentions from dynamic workload, self-adaptation of system software and hardware. The idea of DTiL-RTES is to overcome these by collecting evidence from observations of the system, change the model where it deems it necessary. The decision of whether changing the model is based on outcomes from an internal simulator that evaluates the impact of errors.

In terms of the overhead of this approach, DTiL-RTES has profiling, communication, and memory (for data buffering) overhead. However, most of the heavy computation is offloaded on the Digital Twin host, thus has limited impact on the performance of the client.

3 ETM Modelling and Refinement

We motivated that the ETM is subjected to changes and the DT has the capability to accommodate the issue. However, a number of research questions remain: (1) when it is necessary to change the model; (2) how to change the model; (3) how to evaluate the change to the model is safe. Based on the introduced framework, we aim to provide solutions to (1) and (2), and provides some insights on the third question in this work.

To refine the model at run-time, we introduce a process for ETM error modelling. With a statistical learning process, the error is decomposed and analysed to understand the sufficiency of the relevant model and if the model can be further improved. For troublesome cases, the system would then take a snapshot, record it into a database, before it is further analysed with statistical or machine learning based techniques, for example, with *Principal Component Analysis* (PCA).

The workflow of the approach is shown in Fig. 3. From the figure, the general structure and the flow of the DTiL-RTES approach can be seen in more detail. The basic idea of this approach is to compare the results from simulation using models based on predictions against the models using real observation with an impact analysis.

3.1 Execution Time Model

The Execution Time Model (ETM) is a predictive model that produces execution time estimations based on given inputs. In this work, we apply a model known as cache recency profile (CRP) as it represents a good abstraction of the system. The CRP is a sensitivity model that represents the reduction in WCET against a stress metric, the recency distance, which is the distance measured by cache line accesses between the current job execution and its last instance. This is based on the understanding that execution time is largely dependent on cache reuse. We note that the choice of this model as ETM is just an example use case and we envisage our Digital Twin framework is adaptable to similar predictive

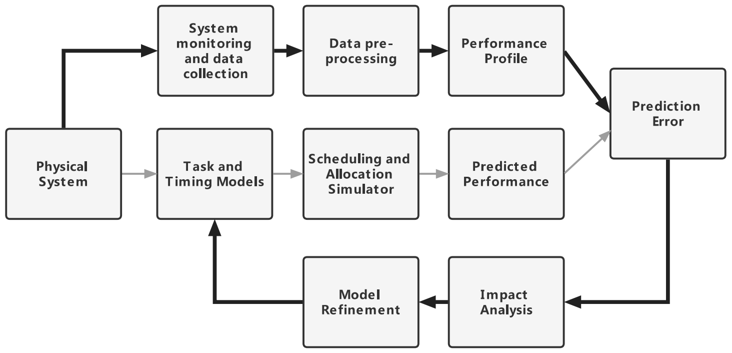

Fig. 3. Workflow of DTiL-RTES for model refinement (bold lines: feedback loop)

timing models, for example, using recurrent neural networks. A full ETM can consist of a combination of models, but in this work we focus on the CRP to illustrate the idea, and use the term ETM and CRP interchangeably.

To clarify, there is a few assumptions of applying this model: (i) application tasks are modelled as functions with a single entry point; (ii) level of task abstraction: each task represents the minimal schedulable entity. Within a node there is no multi-threading. However, the level of abstraction in itself is a research question; (iii) the system has turned off *dynamic voltage and frequency scaling* (DVFS). DVFS adds significant interference to timing and adds unnecessary dynamics that is not favoured against predictability; and (iv) the system uses non-preemptive scheduling, for example, standard Linux OS without RT-patch or any other *commercial off-the-shelf* (COTS) OS. These assumptions form a very common setup of industrial real-time embedded systems.

The CRP is a simple yet effective abstraction of the temporal behaviour of a task. For the current implementation of CRP, the following factors are considered implicitly even if not being directly modelled: (i) prior-condition of tasks execution, (ii) data dependency between tasks, and (iii) instruction dependency through shared libraries.

3.2 Offline Profiling of CRP Model

The initial CRP model is produced offline with the support of a cache analysis tool (Valgrind[3]). Alternatively, it can be generated with static program analysis with control flow graph. The offline profiling tool stretches the independent variable (*i.e.,* the recency distance; defined in Section 3.1), and estimates the corresponding dependent variable (*i.e.,* the ET w.r.t. the percentage of WCET). The process is shown in Fig. 4.

However, the model produced in this way is limited by the prior-assumptions of the system, and the capability of the tool used. For example, the tool has limited support of multi-core, thus is not able to capture multi-core effects. The model will hardly be right, as we motivated earlier. The lack of run-time information, the exact impact factors to the execution time, cannot be fully known until the system is in an operating state. The initial model has to be either

[3] `https://valgrind.org/`

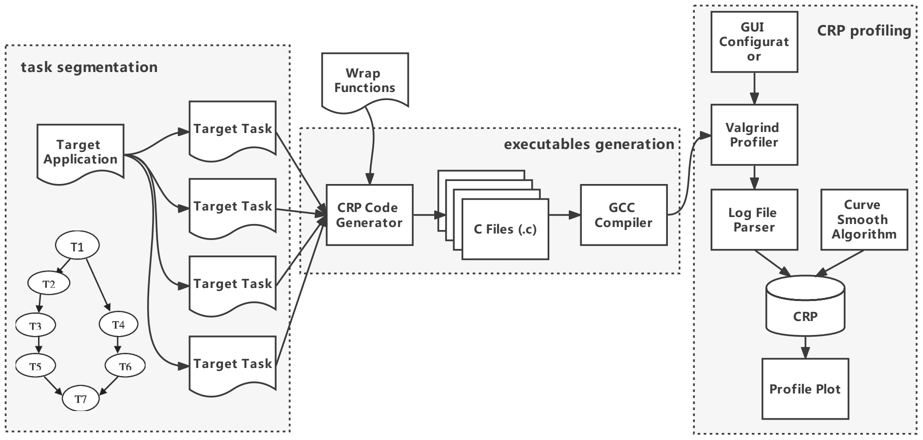

Fig. 4. The offline CRP profiling process

pessimistic or optimistic, or partially pessimistic for some parts but optimistic for the others.

3.3 Prediction Error

It is expected that the offline profiled model is not capable of fully capturing all the features related to task timing, and thus prediction errors are inevitable. A prediction error in this context is defined as the difference between the prediction and the real observation, *i.e.,* $e_i = \hat{y}_i - y_i$. Errors are inevitable in the modelling process and it is important to identify the existence of errors and its impact to the performance. The prediction error can be evaluated using either squared error, $e_i^2 = (\hat{y}_i - y_i)^2$, or root mean squared error (RMSE), $RMSE = \sqrt{\sum (\hat{y}_i - y_i)^2 / n}$.

Generally, there are common sources of errors in the context of a multi-core real-time embedded systems. To list a few:

- Data and instruction dependency across the tasks.
- New tasks arriving and old tasks finishes execution/terminates.
- Change of system modes due to switch of environment.
- Bus interference due to, *e.g.,* memory and I/O access.
- Operating system interference (*e.g.,* Linux services).

These are known unknowns, *i.e.,* we know their existence but do not know when and how they will have impact. Note these interferences will not be considered explicitly but implicitly in the error modelling. While the rest that is not categorised here would be considered as unknown unknowns, *i.e.,* we do not know their existence and do not know when and how they will have impact, and be considered as contributions to the errors.

3.4 Continuous Refinement through Naive Feedback

The model can be improved incrementally through a feedback-based process. In this naive approach, the CRP model is improved through a feedback loop, and we introduce a parameter (L) as the feedback gain. Assuming the current active CRP model to be $CRP = \{c_0, c_1, ..., c_n\}$ where c_i is a model parameter and n is the total number of model parameters (*i.e.,* degree of freedom, or DoF), and the candidate model to be $CRP^c = \{c_0^c, c_1^c, ..., c_n^c\}$. We define a new operator, $\otimes$, as element-wise adaptation. We also define a function, $g(X_1, X_2)$, to extract modelling errors between models represented by parameters X_1 and X_2, respectively. The feedback model update process is defined as:

$$\begin{bmatrix} c_0' \\ c_1' \\ \cdots \\ c_n' \end{bmatrix} = L \otimes g\left(\begin{bmatrix} c_0 \\ c_1 \\ \cdots \\ c_n \end{bmatrix}, \begin{bmatrix} c_0^c \\ c_1^c \\ \cdots \\ c_n^c \end{bmatrix}\right) + \begin{bmatrix} c_0 \\ c_1 \\ \cdots \\ c_n \end{bmatrix} \tag{1}$$

where $L \in [0, 1]$ is the gain, which can be understood as a 'learning rate' and it controls the speed of the adaptation process. The new parameters are the result of the original parameters plus an error matrix that is extracted from the candidate model against the original model. This adaptation method does enable timely changes – depending on the adaptation frequency, a change in the model can take place every few hours or even a few minutes. However, we note this naive approach lacks guarantee on stability. The continuous change could invalid the safety properties if applied without any constraints, and may lead to unnecessary changes and may also be too frequent.

3.5 Model Refinement through Condition-based Model Rebuilding

To overcome the drawbacks of the naive approach, an alternative way of refining the model is through a more controlled process that rebuilds the model based on significance of observations. The idea is that the action needs to pass through an impact identification phase, before the new model can be applied to the system. The intuition behind this strategy is that an error is relevant only when it has an impact on system performance. Any decision that is made to refine the model should come with the expectation that the refinement would lead to improvement of the system.

The whole process is shown in Fig. 5. The input data is firstly collected and cleaned with anomalies being removed. The challenge to do so is to distinguish anomalies from outliers, as outliers may contain information that is important to the modelling process, but anomalies will negatively impact the accuracy thus lead to a unusable model. As an example, we remove anomalies based on statistics of every 200 samples, where any data point that is outside the range of $\mu \pm 3 \times \sigma$ is removed. The anomaly removal can be much more complicated.

The candidate models are then built after a resampling and classification process. The identification of the error impact is through a statistical test against simulated result based on the new and the original timing models. The scenario database (*SCN_DB*) is the database to save identified representative/difficult scenarios which helps to train the model in the future. When testing, the process will also go through the test cases in the scenario database. This overcomes the problem of limited sampling window and avoid considering all historical data.

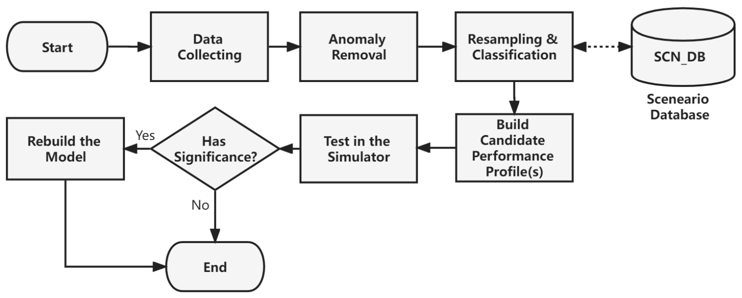

Fig. 5. Flow chart of ETM error modelling

To fit the CRP model, a piece-wise linear regression (PWLR) model is applied for each task:

$$y(x) = \begin{cases} \eta_1 + \beta_1 (x - b_1), & b_1 < x \leqslant b_2 \\ \eta_2 + \beta_2 (x - b_2), & b_2 < x \leqslant b_3 \\ \cdots \\ \eta_n + \beta_\mathrm{n} (x - b_{\mathrm{n}-1}), & b_{\mathrm{n}-1} < x \leqslant b_\mathrm{n} \end{cases} \tag{2}$$

where the parameters of the model (η_x, β_x, b_x) are found using least square estimation that minimised the RMSE. When building the PWLR model, the resampling processing can choose to favour the rare cases more, or average cases more. This is depending on the requirement of the scheduling algorithm. After the CRP model is obtained, there is an associated error evaluation process, in which it can indicate the model is not adequate and thus it needs to be changed (for example, by using a different level of abstraction). This is supported by an internal simulator that simulates scheduling and allocation where it utilises the ETM to estimate the system timing performance.

In DTiL-RTES, there are two levels of assessments, and consequent actions. One level is tuning the parameters of the model (this paper) and the other is to redesign the model (future work). When tuning the parameters, an automatic process is introduced that can adjust the model parameters to fit the observations and thus reduce the error. However in some cases, the tuning would be inadequate and the model may need to be re-designed. We thus compensate this by making humans in the loop, and introduce a testing phase which indicates the existence of such scenarios. Finally, a decision on rebuilding the model is made based on the impact analysis, where statistical test is performed based on the performance evaluated in the high-level simulator.

4 Evaluation

In this section, we evaluate the proposed method with respect to modelling accuracy and improvement in system performance with respect to scheduling results. The evaluation has two main objectives: the first objective is to demonstrate the modelling and error modelling process (in Sec. 4.2); the second objective is to show the impact on the system and how the decision can be made based on the observations (in Sec. 4.3).

#	*Benchmark Task*	*RD*	*MA*	#	*Benchmark Task*	*RD*	*MA*
1	tacle/adpcm_dec	151	306K	8	tacle/ndes	194	127K
2	tacle/adpcm_enc	148	307K	9	tacle/ammunition	970	261G
3	tacle/gsm_dec	536	3.7M	10	tacle/g723_enc	182	1.0G
4	tacle/gsm_enc	916	9.9M	11	tacle/anagram	1215	7.1G
5	tacle/h264_dec	648	402K	12	tacle/audiobeam	1056	1.5G
6	tacle/mpeg2	4105	568M	13	tacle/huff_dec	477	368K
7	tacle/statemate	97	60.6K	14	tacle/huff_enc	840	1.6G

Table 1. Taskset and key parameters used (RD: recency distance; MA: memory access)

4.1 Evaluation setup

The testbed environment uses an Intel Core i5 quad-core processor running at 800 MHz with 16 GB of RAM. Three of the four cores (core 1-3) are used as worker cores to run application tasks, and one core (core 0) is used to run the global scheduler and the Digital Twin client. The dynamic voltage and frequency scaling (DVFS) is disabled. The data was profiled on the client then transferred in chunks to a desktop PC that serves as the Digital Twin host. The client schedules jobs with a global non-preemptive task scheduler, *i.e.,* a job will not be preempted by another job on the same core once it is executed.

The taskset we used is from a well-established benchmark in real-time embedded systems, the *TACLe Benchmarks* [6]. We modelled the taskset with an off-line modelling tool. The main parameters of the tasks are shown in Table 1. The tasks were released randomly, and the system scheduler randomly chose one task to run. The execution time of each task (the dependent variable) and the accumulated recency distance since its last run (the independent variable) was recorded into a light-weight SQL database (SQLite[4]).

4.2 Modelling and Residual Error Evaluation

First, we evaluated the effectiveness of the modelling method and the modelling accuracy. The observation is organised into a (*Recency Distance*, *Execution Time*) pair. We select one of the tasks, *tacle/ndes*, for further analysis. There are overall 24868 valid data points after removal of anomalies, of which 80% is used for training and 20% is used for testing. A piece-wise regression model was then fitted onto the data using least square estimation as described earlier.

The residual error is then evaluated with the prediction from the testing data against the number of model parameters n (shown in Fig. 6) to understand the trade-off between model complexity and sum of squared error. By scaling n from 2 to 12, the error drops significantly for the first iterations but then stops after $n = 7$. This indicates that a more complex model would not always produce a better result, and thus there is clearly an optimal model complexity in the sense that it produces accurate results but at a reasonable low computational and memory cost. The other observation is that from the histograms on the top of the diagram, it can be seen that the error does not follow a normal distribution. There are two modals of the distribution, one at 0 and another at 1. The message it deliveries is that the error pattern indicates the model has accurately captured

[4] https://www.sqlite.org

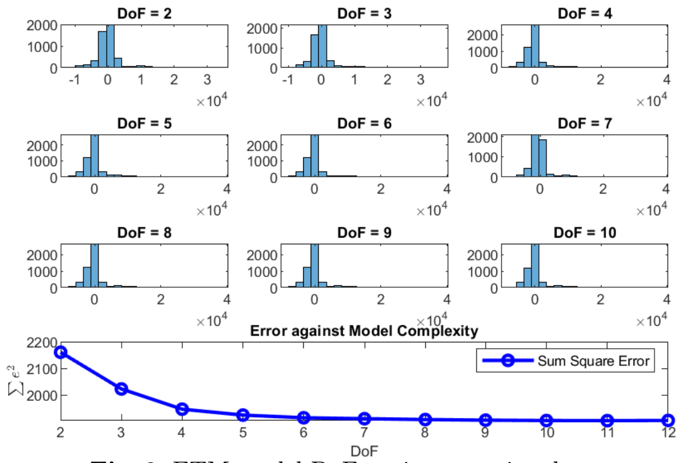

Fig. 6. ETM model DoF against associated error

the main characteristics, however there could be another factor that is missed from the model.

To further investigate, by making $n = 10$, we evaluated the sum of squared residual error, $\sum(y_i - \hat{y_i})^2$ where y_i is from the observation and $\hat{y_i}$ is the estimation based on the predictive model. The error is plotted out against the recency distance. The result is shown in Fig. 7. From the figure, it can be seen that the model fits well with the data in general. However, as can be seen, there is a large number of outliers that lie in the range of 5 - 10 $\times$ 10^4. For dependable systems, these outliers can be more important than the normal data that is successfully captured by the model and should be recorded into the scenario database for further analysis.

4.3 Evaluation of Model Refinement and Performance Improvement

To see how our Digital Twin can be used in decision making and model refinement, we compared the system performance using old (based on offline synthetic model) and new CRPs (based on observations from the real system). The evaluation metric of performance is the total execution time of 100 randomised job instances. The result is shown in Fig. 8. Two pairs of statistical tests were done with a non-parametric test (*Kolmogorov-Smirnov test*, or *K-S test* [11]) applied, with the null hypothesis H_0 being the two datasets are from the same distribution. The system made a decision to apply the new model based on the first test as there exists statistically difference. The decision is further evaluated by applying the new model to the system and re-measured the performance.

The second test between the sim and the real data gave a more positive result, with null hypothesis not being rejected and p-value = 0.307. This shows the predictions from the Digital Twin are broadly similar to the real data. However, this is not a strong accept, and by cross-comparing the data distribution, there are still significant differences and some could affect dependability.

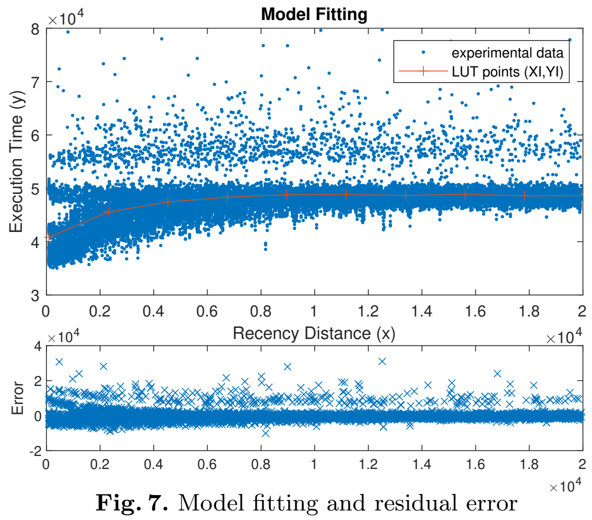

Fig. 7. Model fitting and residual error

4.4 Discussion on safety challenges

As demonstrated in the evaluation, although benefits can be gained, we note safety is important to construct a dependable system. Systems such as communication base stations can be more open for adaptation while the other systems such as those in avionics are less adaptable and more sensitive to risks. The involvement of decision making in mission-critical or safety-critical systems makes it vital to argue the system integrity. The current statistical test lacks richness with respect to safety guarantees, and it requires formal methods and formal verification during the decision making process [18], where model checking tools including *UPPAAL*[5] and *PRISM*[6] can be applied. Concepts such as *Models@runtime* [16] can also be utilised, as well as converting the current safety case from *Goal Structuring Notation* (GSN) [9] to a model-based safety argument, *e.g.,* using *Structured Assurance Case Metamodel* (SACM) [17]. However, as revealed in the evaluation, the outliers and misaligned data distribution, which are keys to safety, could make formal methods much more challenging. Again, this is related to how the model is built and what level of abstraction is sufficient.

A further challenge is also motivated. That is, when it is safe to change the model being used and how to make transition between models. For this challenge, it will be key that rely-guarantee style contracts [2,15] are specified and proven to uphold the essential safety properties of the system. The proof that these are met will be needed both offline for the mechanisms and online for the specific changes applied.

5 Related Work

In this section, we introduce related work from two facets: timing prediction in multi-cores, and execution time modelling for real-time and embedded systems.

[5] `https://uppaal.org/`

[6] `https://www.prismmodelchecker.org/`

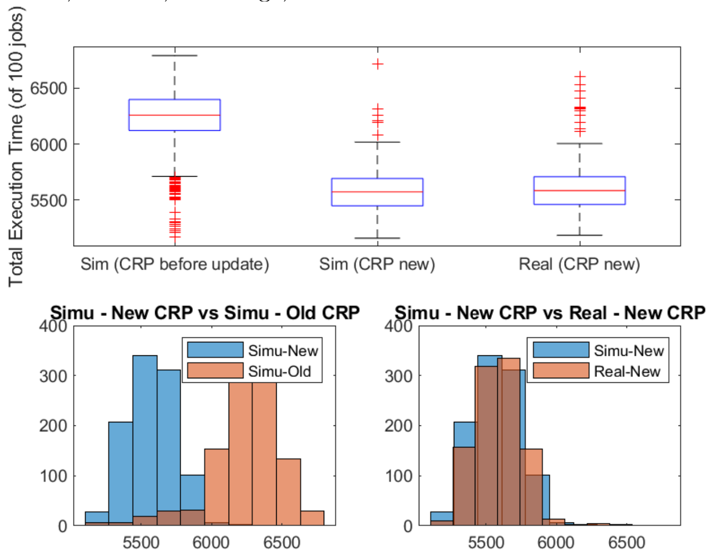

Fig. 8. Comparison of estimation from DTiL-RTES v.s. real observation

5.1 Timing Prediction for Multi-core Real-Time Embedded Systems

Traditionally, when modelling multi-core systems, the integrated timing behaviour is represented by the composition of a number of independent modelled factors including bus interconnection, cache reuse, memory contention, context switches, operating system (OS) interference and so on. However, for COTS systems, it is hard to consider some individual factors – in terms of hardware, some processor specifications are vague, inaccurate and even incorrect; and with respect to software, some binary code and dynamics libraries are closed-source, and even worse when the application is running on an operating systems that is not initially designed for real-time, *e.g.,* Linux, where the characteristics of the timing due to OS is difficult if not impossible to model accurately [12].

Challenges for modelling real-time multi-core systems also lie in the expressiveness of timing. Temporal logical models [14] were used to model the timing behaviour of dynamic systems, however as computer systems exhibit more non-linear characteristics, it is beyond the expressiveness of current modelling capability. The other attempt is to formally model the multi-core system as a group of state machines where each core (or thread) has an independent state diagram with another for synchronisation. This formal analysis is often associated with functional safety, *e.g.,* checking the system is deadlock-free from sharing resources.

5.2 Execution Time Modelling

In the context of modelling of multi-core timing, cache is recognised as one of the most impactful factors. *Static probabilistic timing analysis* (SPTA) and *measurement-based probabilistic timing analysis* (MBPTA) are the two methods for modelling cache for deriving the bound of worst-case execution times [1] [10]. While some of these works consider cache, their purpose is still mainly for deriving the worst-case execution times offline, where these information is not further utilised.

Recent work including analytical cache model with data-driven learning methods provides another direction for execution time modelling, which including *Deep Neural Network* (DNN) and *Recurrent Neural Network* (RNN) including *Long Short-Term Memory* (LSTM). A survey of using some of those models for computer architecture design is given in [13]. There is also work that focuses on inter-core cache effects, *e.g.,* cache interference prediction in multi-cores [8]. Also with high prediction accuracy, those models either does not support run-time use, or has significant overheads for online scheduling and model re-training.

6 Conclusion

In this work, we introduced the DTiL-RTES approach that aims to improve the timing models at run-time to enhance adaptiveness, resilience and robustness of the system. DTiL-RTES makes decisions based on data observed from the system in operation. We first introduced the DTiL-RTES framework, including its key functions and components. We then demonstrated the performance and introduce further challenges. Future work includes how to use this framework to improve scheduling and allocation, apply formal analysis on this framework, and extend the framework with a run-time monitor and anomaly detector.

References

1. Bernat, G., Colin, A., Petters, S.: pWCET: A tool for probabilistic worst-case execution time analysis of real-time systems. University of York (2003)
2. Burns, A., Jones, C.: An approach to formally specifying the behaviour of mixed-criticality systems. In: Euromicro Conference on Real-Time Systems. ACM (2022)
3. Dai, X., Burns, A.: Predicting worst-case execution time trends in long-lived real-time systems. In: Ada-Europe International Conference on Reliable Software Technologies. pp. 87–101. Springer (2017)
4. Dai, X., Burns, A.: Period adaptation of real-time control tasks with fixed-priority scheduling in cyber-physical systems. Journal of Systems Architecture (2020)
5. Dai, X., Zhao, S., Bate, I.J., Burns, A., Guo, X., Chang, W.: Brief industry paper: Digital twin for dependable multi-core real-time systems—requirements and open challenges. In: Real-Time and Embedded Technology and Applications Symposium. IEEE (2021)
6. Falk, H., Altmeyer, S., Hellinckx, P., Lisper, B., Puffitsch, W., Rochange, C., Schoeberl, M., Sørensen, R.B., Wägemann, P., Wegener, S.: Taclebench: A benchmark collection to support worst-case execution time research. In: 16th International Workshop on Worst-Case Execution Time Analysis (2016)
7. Feng, H., Gomes, C., Thule, C., Lausdahl, K., Iosifidis, A., Larsen, P.G.: Introduction to digital twin engineering. In: 2021 Annual Modeling and Simulation Conference (ANNSIM). pp. 1–12. IEEE (2021)
8. Griffin, D., Lesage, B., Bate, I., Soboczenski, F., Davis, R.I.: Forecast-based interference: Modelling multicore interference from observable factors. In: Proceedings of International Conference on Real-Time Networks and Systems (2017)
9. Kelly, T., Weaver, R.: The goal structuring notation–a safety argument notation. In: Proceedings of the dependable systems and networks 2004 workshop on assurance cases. p. 6 (2004)

10. Lesage, B., Griffin, D., Soboczenski, F., Bate, I., Davis, R.I.: A framework for the evaluation of measurement-based timing analyses. In: Proceedings of international conference on real time and networks systems (2015)
11. Massey Jr, F.J.: The kolmogorov-smirnov test for goodness of fit. Journal of the American statistical Association (253), 68–78 (1951)
12. de Oliveira, D.B., Casini, D., de Oliveira, R.S., Cucinotta, T.: Demystifying the real-time linux scheduling latency. In: 32nd Euromicro Conference on Real-Time Systems (ECRTS 2020). Schloss Dagstuhl-Leibniz-Zentrum für Informatik (2020)
13. Penney, D.D., Chen, L.: A survey of machine learning applied to computer architecture design. arXiv preprint (2019)
14. Pnueli, A., Harel, E.: Applications of temporal logic to the specification of real time systems. In: International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems. pp. 84–98. Springer (1988)
15. Sangiovanni-Vincentelli, A., Damm, W., Passerone, R.: Taming dr. frankenstein: Contract-based design for cyber-physical systems. European journal of control **18**(3), 217–238 (2012)
16. Szvetits, M., Zdun, U.: Systematic literature review of the objectives, techniques, kinds, and architectures of models at runtime. Software & Systems Modeling **15**(1), 31–69 (2016)
17. Wei, R., Kelly, T.P., Dai, X., Zhao, S., Hawkins, R.: Model based system assurance using the structured assurance case metamodel. Journal of Systems and Software pp. 211–233 (2019)
18. Weyns, D., Iftikhar, M.U., De La Iglesia, D.G., Ahmad, T.: A survey of formal methods in self-adaptive systems. In: Proceedings of the fifth international c* conference on computer science and software engineering. pp. 67–79 (2012)
19. Yan, J., Zhang, W.: WCET analysis for multi-core processors with shared L2 instruction caches. In: Real-Time and Embedded Technology and Applications Symposium. pp. 80–89. IEEE (2008)