Proceedings Paper:

PRESTO: Predicting System-level Disruptions through Parametric Model Checking

Xinwei Fang
Department of Computer Science
University of York, York, UK
xinwei.fang@york.ac.uk

Radu Calinescu
Department of Computer Science
University of York, York, UK
radu.calinescu@york.ac.uk

Colin Paterson
Department of Computer Science
University of York, York, UK
colin.paterson@york.ac.uk

Julie Wilson
Department of Mathematics
University of York, York, UK
julie.wilson@york.ac.uk

ABSTRACT

Self-adaptive systems are expected to mitigate disruptions by continually adjusting their configuration and behaviour. This mitigation is often reactive. Typically, environmental or internal changes trigger a system response only after a violation of the system requirements. Despite a broad agreement that prevention is better than cure in self-adaptation, proactive adaptation methods are underrepresented within the repertoire of solutions available to the developers of self-adaptive systems. To address this gap, we present a work-in-progress approach for the prediction of system-level disruptions (PRESTO) through parametric model checking. Intended for use in the analysis step of the MAPE-K (Monitor-Analyse-Plan-Execute over a shared Knowledge) feedback control loop of self-adaptive systems, PRESTO comprises two stages. First, time-series analysis is applied to monitoring data in order to identify trends in the values of individual system and/or environment parameters. Next, future non-functional requirement violations are predicted by using parametric model checking, in order to establish the potential impact of these trends on the reliability and performance of the system. We illustrate the application of PRESTO in a case study from the autonomous farming domain.

KEYWORDS

self-adaptive system, proactive adaptation, parametric model checking, non-functional requirements, system verification

1 INTRODUCTION

Self-adaptive systems are expected to mitigate disruptions in complex and uncertain environments by continually analysing, evaluating and adjusting their configurations. To achieve this, their behaviours and operating environments need to be modelled and analysed, often using stochastic models such as queueing networks [19], Markov models [20, 31] and stochastic Petri nets [5]. These models are continually updated to reflect changes observed through monitoring, and used to re-verify compliance with the requirements when needed. In this work, we focus on the use of discrete-time Markov chains (DTMCs) for such purpose and consider disruptions as violations of system-level requirements.

Proactive adaptations that make decisions before disruptions occur have a number of advantages over traditional reactive adaptations, especially when associated with cost [32] or latency [29].

However, many proactive adaptations are triggered by the changes of system parameters rather than the anticipated violation of system-level properties [1, 25]. Due to the complexity and non-linearity of system processes, the impact at system-level caused by these low-level changes may not be obvious. For example, large changes in system parameters not necessarily lead to the violation of requirements but small ones may [18]. Therefore, adaptation decisions triggered by changes in system parameters may result in over or under adaptations.

Fortunately, recent advances in the analysis of *parametric* discrete-time Markov chains allow the efficient verification of system-level requirements using runtime observations of system parameters [9, 15, 22]. Additionally, a growing collection of methods for online learning the parameters of these models ensures that underlying changes or trends in these parameters can be successfully captured [10, 17, 38]. Finally, advances in data analysis allow the confident prediction of observed parameters [1, 26, 27]. Combining these methods can enable self-adaptive systems to *predict and prevent* disruptions before they happen.

Our paper introduces a work-in-progress approach for the prediction of system-level disruptions (PRESTO) through parametric model checking. PRESTO is intended for use in the analysis step of the MAPE-K (Monitor-Analyse-Plan-Execute over a shared Knowledge) feedback control loop [3, 8] of self-adaptive systems. PRESTO allows adaptation decisions to be made based on predicted system-level requirement violations due to degradation in system and/or environmental parameters—a common phenomenon in real-world applications [34, 35] that is underexplored by current research into self-adaptive systems. PRESTO can work in conjunction with methods that mitigate other types of disruptions in self-adaptive systems, e.g., sudden requirement violations due to step changes in system and/or environment parameter values [16, 38].

The rest of paper is structured as follows. In Section 2, we provide a formal definition for Markov models and how to define and evaluate system-level properties. Section 3 gives a running example that will be used throughout this paper. We present PRESTO and its preliminary evaluation in Section 4 and 5, respectively, followed by related work on proactive adaptation in Section 6. Lastly, we conclude the paper with a brief summary in Section 7.

2 PRELIMINARIES

Discrete-time Markov chains (DTMCs) are finite state-transition systems comprising *states* associated with relevant configurations of a system under analysis, and *probabilistic state transitions* that model the stochastic behaviour of that system.

Definition 2.1. A discrete-time Markov chain over a set of atomic propositions AP is a tuple

$$D = (S, s_{init}, M, L), \tag{1}$$

where $S \neq \emptyset$ is a finite set of states; $s_{init} \in S$ is the initial state; $M : S \times S \to [0, 1]$ is a transition probability matrix such that, for any pair of states $s, s' \in S, M(s, s')$ provides a value indicating the probability of transitioning from s to s', and $\sum_{s' \in S} M(s, s') = 1$ for any $s \in S$; $L : S \to 2^{AP}$ is a labelling function that maps every state $s \in S$ to the atomic propositions from AP that hold in that state.

A *reward* can be assigned to states to extend the range of non-functional properties that can be analysed.

Definition 2.2. A reward structure over a discrete-time Markov chain (1) is a function

$$rwd : S \to \mathbb{R}_{\geq 0} \tag{2}$$

that associates non-negative values with the DTMC states.

Parametric discrete-time Markov chains (pDTMCs) can be utilised for the analysis of reward-augmented DTMCs with unknown probabilities and/or rewards.

Definition 2.3. A parametric discrete-time Markov chain (pDTMC) is a discrete-time Markov chain (1), with or without a set of reward functions (2), that includes transition probabilities $P(s, s')$ and/or rewards specified as rational functions over a set of real-valued parameters.

Probabilistic computation tree logic (PCTL) [2, 7, 21] is a probabilistic variant of temporal logic used to specify the properties of DTMCs.

Definition 2.4. A PCTL *state formula* Φ, *path formula* Ψ, and *reward state formula* Φ_R over an atomic proposition set AP are defined by:

$$\Phi ::= true \mid a \mid \neg\Phi \mid \Phi \wedge \Phi \mid \mathcal{P}_{=?}[\Psi] \tag{3}$$

$$\Psi ::= X\Phi \mid \Phi \, U^{\leq k} \, \Phi \mid \Phi \, U \, \Phi \tag{4}$$

$$\Phi_R ::= \mathcal{R}_{=?}^{rwd}[I^{=k}] \mid \mathcal{R}_{=?}^{rwd}[C^{\leq k}] \mid \mathcal{R}_{=?}^{rwd}[F \, \Phi] \mid \mathcal{R}_{=?}^{rwd}[S] \tag{5}$$

where $a \in AP$ is an atomic proposition, $k \in \mathbb{N}_{>0}$ is a timestep bound, and rwd is a reward structure.

The PCTL semantics is defined using a satisfaction relation $\models$ over the states $s \in S$ and paths $\pi \in Paths^D(s)$ of a DTMC. Thus, $s \models \Phi$ means "Φ holds in state s", $\pi \models \Psi$ means "Ψ holds for path π", and we have: $s \models true$ for all states $s \in S$; $s \models a$ iff $a \in L(s)$; $s \models \neg\Phi$ iff $\neg(s \models \Phi)$; $s \models \Phi_1 \wedge \Phi_2$ iff $s \models \Phi_1$ and $s \models \Phi_2$. The quantitative state formula $\mathcal{P}_{=?}[\Psi]$ specifies the probability that paths from $Paths^D(s)$ satisfy the path property Ψ. *Reachability properties* $\mathcal{P}_{=?}[true \, U \, \Phi]$ are equivalently written as $\mathcal{P}_{=?}[F \, \Phi]$ or $\mathcal{P}_{=?}[F \, R]$, where $R \subseteq S$ is the set of states in which Φ holds. The *next formula* $X\Phi$ holds if Φ is satisfied in the next state of the analysed path π. The *time-bounded*
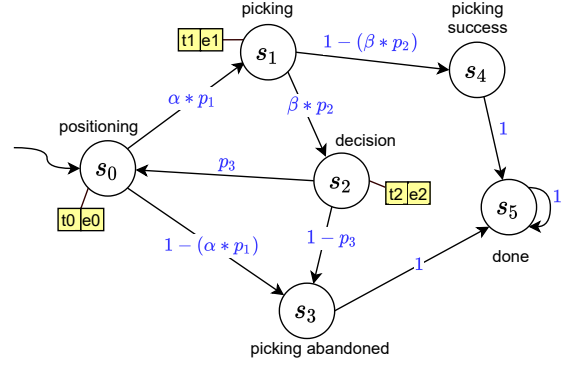

Figure 1: pDTMC model of autonomous fruit-picking process

until formula $\Phi_1 \, U^{\leq k} \, \Phi_2$ holds for a path π iff $\pi(i) \models \Phi_2$ for some $i \leq k$ and $\pi(j) \models \Phi_1$ for all $j = 1, 2, \ldots, i - 1$. The *unbounded until formula* $\Phi_1 \, U \, \Phi_2$ removes the bound k from the time-bounded until formula. Finally, the reward state formulae specify the expected values for: the *instantaneous reward* at timestep k ($\mathcal{R}_{=?}^{rwd}[I^{=k}]$); the *cumulative reward* up to timestep k ($\mathcal{R}_{?}^{rwd}[C^{\leq k}]$); the *reachability reward* cumulated until reaching a state that satisfies a property Φ ($\mathcal{R}_{=?}^{rwd}[F \, \Phi]$); The *steady-state reward* in the long run ($\mathcal{R}_{=?}^{rwd}[S]$). For detailed descriptions of the PCTL semantics, see [2, 7, 21].

Parametric model checking (PMC) [13] is a mathematically based technique for the verification of PCTL-encoded pDTMC properties. Supported by probabilistic model checkers such as PRISM [23] and Storm [14], the technique yields rational functions (i.e., a quotients between two polynomials with respect to the parameters) for the analysed PCTL property. These functions, which we term *PMC expressions* in the remainder of the paper, can be efficiently evaluated by a self-adaptive system at runtime, when the value of the pDTMC parameters are observed through the MAPE-K monitoring.

3 MOTIVATING EXAMPLE

We motivate PRESTO and demonstrate its effectiveness for a fruit-picking robot from the autonomous farming domain. Our example, inspired by recent development in this domain [36, 37], assumes that the robot needs to perform three operations autonomously: (1) position itself in the right location for the fruit picking; (2) use its arm to pick up the fruit; and (3) when operation 2 is unsuccessful, decide whether to retry the fruit picking from operation 1 or to give up. Fig. 1 shows the pDTMC model of this process. The robot begins with positioning itself next to a piece of fruit (state s_0). The positioning operation may succeed, in which case the robot moves to state s_1, where it attempts picking, or may fail, in which case the picking will be abandoned (s_3) and the process finishes (s_5). The process also finishes when the picking is successful (s_4). If the picking is unsuccessful, the robot enters the decision state (s_2) from where it needs to decide whether to re-position itself and retry the entire process, or to abandon picking this fruit (s_3) and end the process.

As shown in Fig. 1, the outgoing transition probabilities from states s_0, s_1 and s_2 are defined in terms of several system parameters. Out of these parameters, we assume that p_1, p_2 and p_3 have

Table 1: System-level requirements, related properties, and the result of parametric model checking for the properties

	Informal description	Property to check (PCTL)	PMC expressions†
R1	The robot shall complete the fruit picking successfully with probability of at least 0.8	$\mathcal{P}_{=?}[F\text{ "picking success"}] \geq 0.8$	$(\alpha * p1 * \beta * p2 + (-1) * \alpha * p1)/(\alpha * p1 * \beta * p2 * p3 + (-1))$
R2	The expected time to complete the picking process shall not exceed 30 seconds.	$\mathcal{R}_{=?}^{\text{"time"}}[F]\text{"done"}] \leq 30s$	$(-1 * (\alpha * p1 * \beta * p2 * t3 + t1 + \alpha * p1 * t2))/(\alpha * p1 * \beta * p2 * p3 + (-1))$
R3	The expected energy consumption to complete the picking process shall not exceed 10 joules.	$\mathcal{R}_{=?}^{\text{"energy"}}[F]\text{"done"}] \leq 10J$	$(-1 * (\alpha * p1 * \beta * p2 * e3 + e1 + \alpha * p1 * e2))/(\alpha * p1 * \beta * p2 * p3 + (-1))$

† PMC expressions were returned from the model checker Storm [14] by providing the pDTMC and the property specified in PRISM language [23] and PCTL respectively.

predefined, domain-specific values. For instance, p_1 is the probability of successful positioning when the robot is in perfect working condition. In contrast, $\alpha, \beta \in (0, 1]$ represent coefficients that reflect the robot degradation from its perfect working condition, e.g. due to wear and tear. We assume that α and β are unknown, and their values need to be obtained through monitoring [24] or via self-testing [30].

Additionally, the pDTMC is annotated with two reward functions (depicted in rectangular boxes linked to states s_0, s_1 and s_2 from Fig. 1). First, a "time" reward function associates mean operation execution times t_0, t_1 and t_2 with the three operations performed by the robot. Similarly, an "energy" reward function associates mean energy consumptions e_0, e_1 and e_2 with the same operations. We assume that the values of t_i and e_i, $0 \leq i \leq 2$, are unknown and need to be obtained through monitoring.

Finally, we assume that the autonomous robot has to satisfy the three system-level requirements shown in Table 1.

4 PRESTO APPROACH

The PRESTO analysis process, depicted in Fig. 2, takes as input a pDTMC model of the self-adaptive system's behaviour, a set of PCTL-encoded system requirements, a *prediction horizon h > 0*, and runtime observations of the unknown system and/or environment parameters that appear in the pDTMC. The first three inputs are provided, at design time, by system engineers and domain experts,

whereas the runtime observations are obtained from the monitoring component of the MAPE-K control loop.

Given these inputs, PRESTO predicts violation of the system-level requirements *within the prediction horizon h*. To that end, our approach first uses a parametric verification engine to obtain PMC expressions for the PCTL properties associated with the system requirements. Typically, this is a one-off step executed at design time; however, PRESTO also allows the run-time execution of this step, if needed because the model structure and/or the system requirements change in operation. The PMC expressions are then used at runtime, to continually assess the impact of predicted parameter changes on the system-level requirements. In this step, PRESTO uses recent observations of the parameter values to predict the parameter future values within the prediction horizon. These predicted parameter values are then "propagated" through the PMC expressions (i.e., they are used to evaluate the PMC expressions) in order to predict the future values of the nonfunctional properties from the system requirements, and thus to predict future requirement violations.

5 EVALUATION

A prototype of PRESTO was implemented to aid evaluation and answer the following research questions:

RQ1 Can PRESTO predict the violation of system-level disruptions and how does it perform in terms of prediction errors?
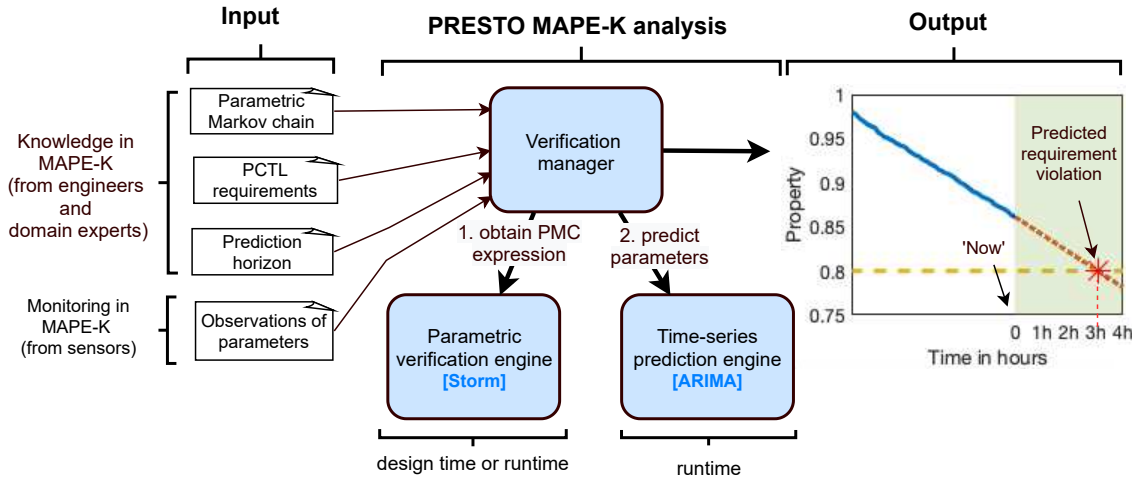
Figure 2: PRESTO analysis process and architecture

RQ2 How does the noise level in model parameters affect PRESTO's prediction performance?

RQ3 How can we utilise PRESTO for proactive adaptation?

The prototype and a simple simulator of the autonomous robot from our motivating example were implemented in Java, using a modular design as shown in Fig. 2. The modular design enables: 1) technologies used in this version of PRESTO to be replaced with alternatives if deemed more suitable for a given application and 2) additional technologies can be integrated to extend the functionality or to improve efficiency. This early version of PRESTO uses the following technologies:

- The parametric verification engine uses Storm [14], one of the leading parametric model checkers, in the background. As an alternative, fPMC [15] is worth exploring, especially when the system under analysis is complex and with many parameters.
- The time-series prediction engine uses ARIMA (Autoregressive Integrated Moving Average), one of the most widely used methods for time-series prediction, also invoked in the background. Numerous alternative prediction methods could be substituted for this module depending on requirements.

In our evaluation, we assigned values $p1 = 0.95$, $p2 = 0.2$ and $p3 = 0.95$, and gave ranges and trends for the other parameters as shown in Table 3. The range and value of model parameters were selected to ensure that the system-level properties would vary around the requirement, but can be based on prior knowledge in practice. Calculations were performed on a Macbook Pro with 2.3GHz Quad-core i7 CPU and 32 GB of RAM and all "monitored" parameters were updated every minute with a prediction horizon set for 240 minutes.

RQ1: To answer this research question, we predicted the violation of a system-level requirement by propagating the predicted parameters within the prediction horizon via the PMC expressions presented in Table 1. We first present one experiment in detail for illustration of the prediction results, and then we analyse the result from a large set of experiments. The detailed results are shown in Fig. 3 where the x-axis shows the time in hours and the y-axis indicates the system-level property, for example, in Fig. 3a, the

probability of completing the process with picking being successful. The bound from the requirement is shown as an orange dashed line. The ground truth for the system-level property is plotted as a solid blue line with predicted values as a dotted red line. The prediction horizon is highlighted in green with zero hours indicating the start. It can be seen that PRESTO is able to predict the violation of system-level requirements if there is any (R1 and R2), and predict the trend of the system level property if no violation is occurred within the prediction horizon (R3). The deterministic values of each monitored parameter at different time point are presented in Table 2 to show the tends. We evaluate the performance of PRESTO further by repeating the above experiment for 3000 times with randomly generated trends according to the specifications shown in Table 3. For each experiment, we randomly chose two values that within the specified range for each parameter, and generated a set of values(600 value points for each parameters) that vary randomly between the two values. The unique trends of each parameter was created by sorting the value set in a descending or ascending order according to the specification in Table 3. The first 360 values were used for parameter analysis and the last 240 values were used as the ground truth of prediction horizon. This evaluation can synthesise a large number of scenarios, including the violation of requirements before the prediction (1761 cases for R1, 1600 cases for R2, and 2209 cases for R3 out of 3000), violation of requirement within the prediction horizon (488 for R1, 715 for R2,and 501 for R3), and no violation of requirements during the period of analysis (751 for R1, 685 for R2, and 290 for R3), to minimise the impact of stochasticity in the experiments. The overall time span for a single run including prediction of time-series and evaluating the result is under one second, and the violation requirements can be observed uniformly across the prediction horizon. It is worth noting that during the 3000 runs of evaluating all three requirements, we did not observe any false positives (i.e., the prediction of a violation when one does not occur within the prediction horizon) or false negatives (i.e., a violation within the prediction horizon that was not predicted), which can be due to the combination effect of parameter trends, prediction horizon, and model structure. The violation of requirements before the prediction can be addressed by a reactive adaption method [10] which is not in the scope of this work. We evaluated prediction
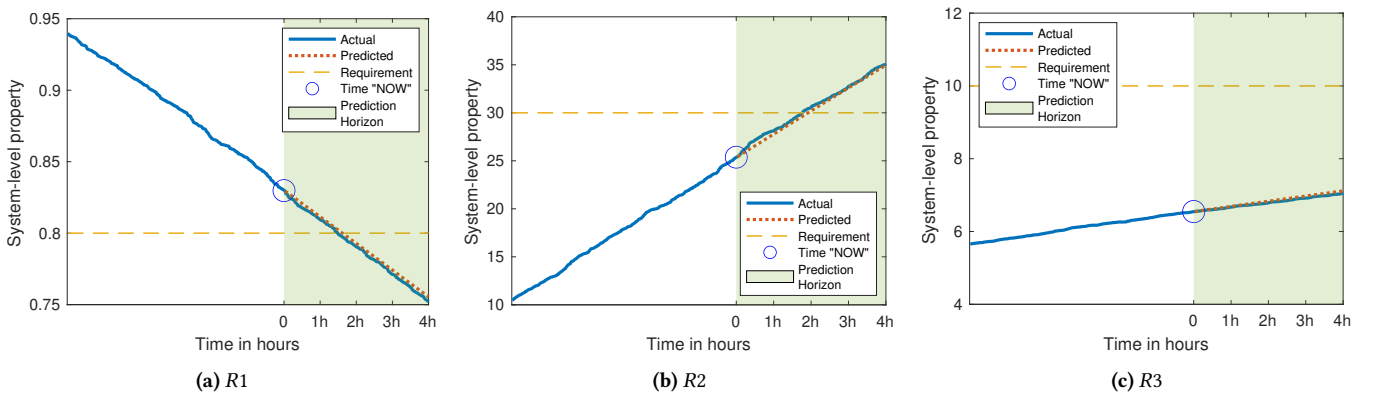

Figure 3: Prediction of system-level properties and potential disruptions for the properties listed in Table 1, the actual disruptions happen at 89-min (R1) and 105-min(R2) in contrast to the predicted disruptions are at 97-min(R1) and 115-min(R2), respectively

(a) *R1*

(b) *R2*

(c) *R3*
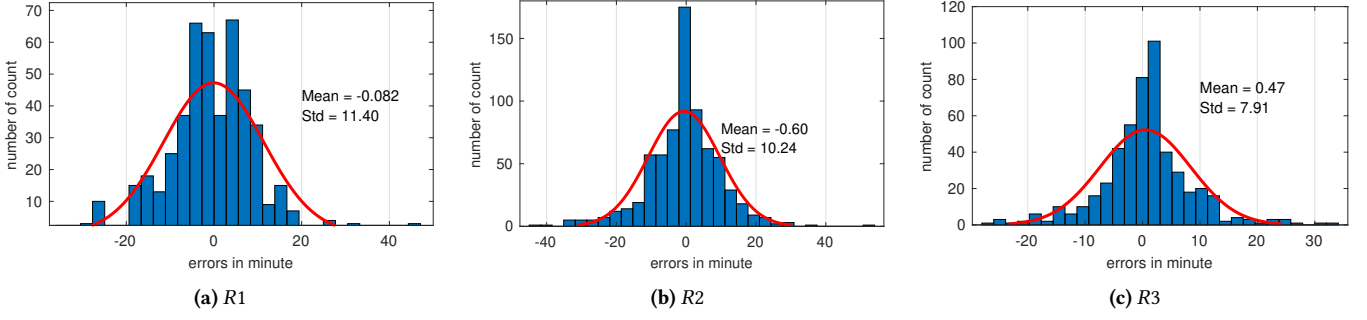
Figure 4: Errors of predicting violation of requirements collected from experiments that have violation of requirements within the prediction horizon

(a) *R1*

(b) *R2*

(c) *R3*
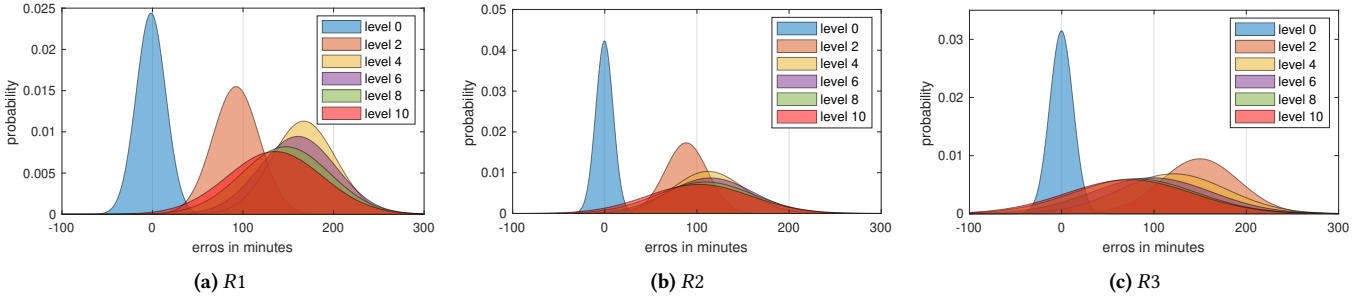
Figure 5: Changes of error distributions as the results of noise level in parameters

(a) *R1*

(b) *R2*

(c) *R3*
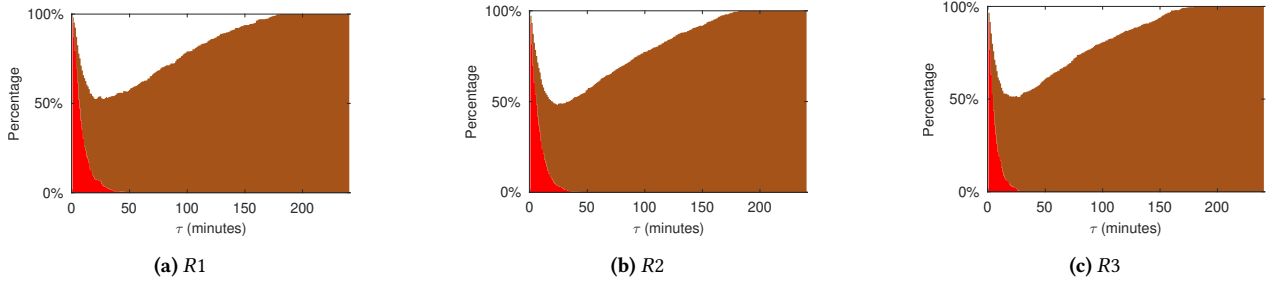
Figure 6: Percentage of undesired cases as a function of τ (out of 488 cases for R1, 715 for R2, and 501 for R3)

errors, calculated as the difference in time between the predicted and actual violation of the requirement, for those experiments that have violation of requirement within the prediction horizon. The histograms together with fitted distributions are shown in Fig. 4. Our preliminary results show the prediction errors are normally distributed with mean values between -0.6 and 0.47 minutes (for R2 and R3 respectively), showing no real bias. The standard deviations of between 7.91 (for R3) and 11.4 (for R1) suggest that PRESTO is likely to predict system-level disruptions accurately.

RQ2: To evaluate the impact of noise on the prediction of disruptions, we added different levels of noise to the generated parameter trends using normal distributions with zero mean but different standard deviations as shown in Table 4. The maximum value of α and β are capped to one, and the minimum values for $t_1, t_2 and t_3$

Table 2: Deterministic parameter values at three time points

	α	β	t_0	t_1	t_2	e_0	e_1	e_2
now-360	0.98	0.01	1.04	10.01	8.90	3.30	2.40	0.30
now	0.88	0.12	11.6	14.6	11.6	4.03	2.77	2.84
now+240	0.80	0.19	19.8	17.9	13.9	4.49	2.99	4.49

Table 3: Ranges and trends for the model parameters

	α	β	t_0, t_1, t_2	e_0, e_1, e_2
range	[0.7,0.99]	[0.01, 0.2]	[1s, 30s]	[0.3J, 4.5J]
trend	constant or monotonically increasing	constant or monotonically decreasing		

Table 4: Standard deviations of noise levels from RQ3

	level 0	level 2	level 4	level 6	level 8	level 10
α, β	0	0.02	0.04	0.06	0.08	0.1
t_0, t_1, t_2	0	2	4	6	8	10
e_0, e_1, e_2	0	0.6	1.2	1.8	2.4	3

and $e_1, e_2, and e_3$ are set to be 1s and 0.3J, respectively. The experiment was repeated 1000 times for each noise level to obtain their distributions as shown in Fig. 5. The distribution with zero mean (in blue), observed in all three cases, is obtained using trends with no added noise (level 0) and provides a baseline. As noise is added, the mean error shifts to the right due to noise spikes violating the requirement before the predicted value, which only considers violations caused by the underlying trend, here increasing or decreasing monotonically. The noise level also impacts the error variance, becoming larger as the noise level increases. These early stage results suggest PRESTO may be sensitive to noise, and that the integration of a de-noising module could be worthwhile.

RQ3: While our preliminary evaluation does not cover the end-to-end use of PRESTO within the MAPE-K loop of self-adaptive systems, we suggest how PRESTO predictions can be exploited by the planning step of the MAPE-K loop. It is clear that adaptation decisions need to be triggered at an appropriate time. In our example, the robot can trigger the decision any time between *now* and predicted violation of requirements. We use a user specified variable, τ, to determine the time to trigger the adaptation decision as $\max(now, (t_p - \tau))$, where t_p is the predicted time that the violation will occur. If τ is too large, the value $(t_p - \tau)$ could represent a time point in the past, in which case, the robot would need to make a reactive adaptation at *now*, possibly generating additional adaptation cost. In addition, a large value of τ could lead to frequent unnecessary adaptations that affect the performance of the robot. On the other hand, if τ is too small, particularly when smaller than the prediction errors, the violation is more likely to happen before being predicted, resulting in high cost from recovery. To facilitate the selection of τ, we illustrate the impact of this parameter in terms of the percentage of undesired cases out of 488 for R1, 715 for R2 and 501 for R3 that could occur using the same experimental data used for RQ1 (Fig. 6). Here $(t_p - \tau) \leq 60min$ is considered too large and $\tau < abs(t_p - t_{ref})$ is considered too small, where t_{ref} is the time of the actual violation of the requirement. The three plots in Fig. 6 show a consistent trend in that the number of undesirable cases is extremely high (up to 100%) when τ is either close to zero or close to its maximum value. Our results show that τ has a sweet spot around 30-min as the percentage of undesired cases goes down as low as 50%. However, if the violation of requirements is associated with a significant cost, a slightly larger τ is preferred.

6 RELATED WORK

Self-adaptive systems can proactively or reactively adapt to changes [6]. While both adaptation techniques have shown good results in handling changes and uncertainties [4], proactive adaptation is particularly important when restoring compliance with requirements after they were violated incurs high costs [32] or latency [29].

Proactive adaptation relies on the prediction of current execution and environmental parameters to identify potential violations and to apply adaptations before these violations occur. While some research in this area concentrated on the improvement of prediction accuracy [12, 27, 28], others investigated possible limitations, such as the trade-off between early prediction and accuracy [26], the impact of adaptation latency [11] and the impact of limited computational resources [33].

Related to PRESTO, the research from [29] integrates probabilistic model checking within a latency-aware proactive adaptation method to determine the best adaptation strategy. The resulting method allows optimal adaptation decisions to be made over the prediction horizon with consideration of the stochastic behaviour of the environment. While this method can be applied at runtime, the entire model needs to be updated when the changes are observed. In contrast, our approach utilises parametric model checking, in which the same PMC expression can be re-used, with only its re-evaluation for new parameter values is required at runtime.

7 CONCLUSION

We have presented a new approach for the prediction of system-level disruptions (PRESTO) which is intended both for use in the analysis step and to support the planning step in the MAPE-K loop. PRESTO uses parametric model checking to establish functional relationships between system and/or environmental parameters and the system-level property. Time-series analysis is used to identify trends in the parameters from which values can be predicted and then propagated via the PMC expressions to assess their impact on system-level properties and to identify potential disruptions. Our preliminary evaluation suggests that PRESTO can accurately predict the system-level disruptions with no false positives or false negatives observed in our experiments.

The benefit of a modular architecture means that the techniques used in PRESTO can be substituted with alternatives according to needs and new technologies can be added to extend functionality. We plan to integrate a de-noising module in future work as evaluation showed noise to have significant impact on the prediction results. Furthermore, more detailed evaluations need to be carried out with different pDTMCs and parameter trends to identify the limitations of PRESTO.

REFERENCES

[1] Ayman Amin, Lars Grunske, and Alan Colman. An automated approach to forecasting qos attributes based on linear and non-linear time series modeling. In *2012 Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, pages 130–139. IEEE, 2012.

[2] Suzana Andova, Holger Hermanns, and Joost-Pieter Katoen. Discrete-time rewards model-checked. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 88–104. Springer, 2003.

[3] Paolo Arcaini, Elvinia Riccobene, and Patrizia Scandurra. Modeling and analyzing mape-k feedback loops for self-adaptation. In *2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 13–23. IEEE, 2015.

[4] Davide Arcelli. Exploiting queuing networks to model and assess the performance of self-adaptive software systems: a survey. *Procedia Computer Science*, 170:498–505, 2020.

[5] Simonetta Balsamo, Peter G Harrison, and Andrea Marin. Methodological construction of product-form stochastic petri nets for performance evaluation. *Journal of Systems and Software*, 85(7):1520–1539, 2012.

[6] Matthias Becker, Markus Luckey, and Steffen Becker. Model-driven performance engineering of self-adaptive systems: a survey. In *Proceedings of the 8th international ACM SIGSOFT conference on Quality of Software Architectures*, pages 117–122, 2012.

[7] Andrea Bianco and Luca de Alfaro. Model checking of probabilistic and nondeterministic systems. In P. S. Thiagarajan, editor, *Foundations of Software Technology and Theoretical Computer Science*, volume 1026 of *LNCS*, pages 499–513. Springer Berlin Heidelberg, 1995.

[8] Yuriy Brun, Giovanna Di Marzo Serugendo, Cristina Gacek, Holger Giese, Holger Kienle, Marin Litoiu, Hausi Müller, Mauro Pezzè, and Mary Shaw. Engineering self-adaptive systems through feedback loops. In *Software engineering for self-adaptive systems*, pages 48–70. Springer, 2009.

[9] R. Calinescu, C. A. Paterson, and K. Johnson. Efficient parametric model checking using domain knowledge. *IEEE Transactions on Software Engineering*, PP:1–1, 2019.

[10] Radu Calinescu, Yasmin Rafiq, Kenneth Johnson, and Mehmet Emin Bakır. Adaptive model learning for continual verification of non-functional properties. In *Proceedings of the 5th ACM/SPEC international conference on Performance engineering*, pages 87–98, 2014.

[11] Javier Cámara, Gabriel A Moreno, and David Garlan. Stochastic game analysis and latency awareness for proactive self-adaptation. In *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 155–164, 2014.

[12] Javier Cámara, Henry Muccini, and Karthik Vaidhyanathan. Quantitative verification-aided machine learning: A tandem approach for architecting self-adaptive iot systems. In *2020 IEEE International Conference on Software Architecture (ICSA)*, pages 11–22. IEEE, 2020.

[13] Conrado Daws. Symbolic and parametric model checking of discrete-time markov chains. In *International Colloquium on Theoretical Aspects of Computing*, pages 280–294. Springer, 2004.

[14] Christian Dehnert, Sebastian Junges, Joost-Pieter Katoen, and Matthias Volk. A storm is coming: A modern probabilistic model checker. In Rupak Majumdar and Viktor Kunčak, editors, *Computer Aided Verification*, pages 592–600, Cham, 2017. Springer International Publishing.

[15] Xinwei Fang, Radu Calinescu, Simos Gerasimou, and Faisal Alhwikem. Fast parametric model checking through model fragmentation. In *Proceedings of the 43rd International Conference on Software Engineering (To appear)*, 2021.

[16] Antonio Filieri, Carlo Ghezzi, and Giordano Tamburrelli. A formal approach to adaptive software: continuous assurance of non-functional requirements. *Formal Aspects of Computing*, 24(2):163–186, 2012.

[17] Antonio Filieri, Lars Grunske, and Alberto Leva. Lightweight adaptive filtering for efficient learning and updating of probabilistic models. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 1, pages 200–211. IEEE, 2015.

[18] Kathi Fisler, Shriram Krishnamurthi, Leo A Meyerovich, and Michael Carl Tschantz. Verification and change-impact analysis of access-control policies. In *Proceedings of the 27th international conference on Software engineering*, pages 196–205, 2005.

[19] G. Franks, T. Al-Omari, M. Woodside, O. Das, and S. Derisavi. Enhanced modeling and solution of layered queueing networks. *IEEE Transactions on Software Engineering*, 35(2):148–161, 2009.

[20] Simos Gerasimou, Radu Calinescu, and Giordano Tamburrelli. Synthesis of probabilistic models for quality-of-service software engineering. *Automated Software Engineering*, 25(4):785–831, 2018.

[21] Hans Hansson and Bengt Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512–535, September 1994.

[22] Nils Jansen, Florian Corzilius, Matthias Volk, Ralf Wimmer, Erika Ábrahám, Joost-Pieter Katoen, and Bernd Becker. Accelerating parametric probabilistic verification. In *11th International Conference on Quantitative Evaluation of Systems (QEST)*, pages 404–420, 2014.

[23] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *23rd International Conference on Computer Aided Verification (CAV)*, pages 585–591, 2011.

[24] Guozhi Li, Fuhai Zhang, Yili Fu, and Shuguo Wang. Joint stiffness identification and deformation compensation of serial robots based on dual quaternion algebra. *Applied Sciences*, 9(1):65, 2019.

[25] Andreas Metzger, Rod Franklin, and Yagil Engel. Predictive monitoring of heterogeneous service-oriented business networks: The transport and logistics case. In *2012 Annual SRII Global Conference*, pages 313–322. IEEE, 2012.

[26] Andreas Metzger, Tristan Kley, and Alexander Palm. Triggering proactive business process adaptations via online reinforcement learning. In *International Conference on Business Process Management*, pages 273–290. Springer, 2020.

[27] Andreas Metzger, Adrian Neubauer, Philipp Bohn, and Klaus Pohl. Proactive process adaptation using deep learning ensembles. In *International Conference on Advanced Information Systems Engineering*, pages 547–562. Springer, 2019.

[28] Andreas Metzger, Christian Reinartz, and Klaus Pohl. Incremental verification of complex event processing applications for system monitoring. In *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 293–297. IEEE, 2018.

[29] Gabriel A Moreno, Javier Cámara, David Garlan, and Bradley Schmerl. Proactive self-adaptation under uncertainty: a probabilistic model checking approach. In *Proceedings of the 2015 10th joint meeting on foundations of software engineering*, pages 1–12, 2015.

[30] Antonis Paschalis and Dimitris Gizopoulos. Effective software-based self-test strategies for on-line periodic testing of embedded processors. *IEEE Transactions on Computer-aided design of integrated circuits and systems*, 24(1):88–99, 2004.

[31] Colin Paterson and Radu Calinescu. Observation-enhanced QoS analysis of component-based systems. *IEEE Transactions on Software Engineering*, 46(5):526–548, 2018.

[32] Vahe Poladian, David Garlan, Mary Shaw, Mahadev Satyanarayanan, Bradley Schmerl, and Joao Sousa. Leveraging resource prediction for anticipatory dynamic configuration. In *First International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2007)*, pages 214–223. IEEE, 2007.

[33] Federico Quin, Danny Weyns, Thomas Bamelis, Sarpreet Singh Buttar, and Sam Michiels. Efficient analysis of large adaptation spaces in self-adaptive systems using machine learning. In *2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 1–12. IEEE, 2019.

[34] Matthew B Russell, Evan M King, Chadwick A Parrish, and Peng Wang. Stochastic modeling for tracking and prediction of gradual and transient battery performance degradation. *Journal of Manufacturing Systems*, 59:663–674, 2021.

[35] Chaitanya Sankavaram, Bharath Pattipati, Anuradha Kodali, Krishna Pattipati, Mohammad Azam, Sachin Kumar, and Michael Pecht. Model-based and data-driven prognosis of automotive and electronic systems. In *2009 IEEE International Conference on Automation Science and Engineering*, pages 96–101. IEEE, 2009.

[36] Nikolaus Wagner, Raymond Kirk, Marc Hanheide, Grzegorz Cielniak, et al. Efficient and robust orientation estimation of strawberries for fruit picking applications. 2021.

[37] Ya Xiong, Yuanyue Ge, and Pål Johan From. An improved obstacle separation method using deep learning for object detection and tracking in a hybrid visual control loop for fruit picking in clusters. *Computers and Electronics in Agriculture*, 191:106508, 2021.

[38] Xingyu Zhao, Radu Calinescu, Simos Gerasimou, Valentin Robu, and David Flynn. Interval change-point detection for runtime probabilistic model checking. In *2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 163–174. IEEE, 2020.