# Efficient Formal Verification with Confidence Intervals

Naif Alasmari
*Department of Computer Science*
*University of York*
York, U.K.
email: nnma500@york.ac.uk

Radu Calinescu
*Department of Computer Science*
*University of York*
York, U.K.
email: radu.calinescu@york.ac.uk

***Abstract*—Formal verification with confidence intervals is a model checking technique that computes confidence intervals for parametric Markov model properties when observations of the unknown transition probabilities of these models are available. However, the high computational costs of the technique limit its scalability severely. To address this limitation, we introduce efficient formal verification with confidence intervals (eFACT), a model checking tool that enables the efficient analysis of parametric discrete-time Markov chains. eFACT supports the verification of reliability, performance, and other non-functional requirements for larger systems than currently possible. To that end, eFACT integrates recent advances in parametric model checking into a previous tool for formal verification with confidence intervals, and employs an efficient binary search technique to further speed up the determination of the highest confidence level at which a non-functional requirement can be deemed violated or satisfied.**

***Keywords*—*confidence intervals; formal verification; non-functional software requirements; probabilistic model checking.***

## I. INTRODUCTION

Over the years, quantitative verification has been a powerful means for analysing the performance, reliability, and other non-functional properties of systems. However, the analysed system should be modelled carefully and accurately as a Markov model in order to obtain a precise verification result. Building a Markov model for the system is a time-consuming task because it requires determining the system's states and the transitions between them, as well as their probabilities. Establishing the precise probabilities of transitions is challenging [1] since the probabilities can only be estimated, with error margins, from run-time observations of the system, system logs, or based on input obtained from domain experts. Therefore, the error values of probabilities estimation could be accumulated by quantitative verification and can produce inaccurate outcomes due to the non-linearity of Markovian models. The formal verification with confidence intervals (FACT) [1] resolves this limitation by providing an interval for the verification result rather than a single value [2].

FACT is a probabilistic model checker that calculates confidence intervals for properties of parametric Markov chains that have observations for unknown transition probabilities. The current FACT version invokes PRISM [3] to get the algebraic expression for the targeted property of a parametric Discrete-Time Markov Chain (pDTMC) model. However, when the algebraic expression is too large to analyse (i.e., it exceeds the memory or computational resources available), or the behaviour of the parametric model is complex (e.g., has continual change), the capability of FACT becomes limited. Thus, FACT does not scale well to large pDTMCs.

To extend the capability of FACT, our paper introduces $e$fficient $F$ormal verific$A$tion with $C$onfidence in$T$ervals (eFACT), a new model checker that is able to calculate such confidence intervals for larger pDTMC models. eFACT exploits efficient parametric model checking (ePMC), which uses domain-specific modelling patterns [4] in order to produce sets of closed-form subexpressions of the analysed properties, and uses these subexpressions as terms in the main formula for the analysis of the whole pDTMC model. The ePMC derives an abstraction model from the original pDTMC. The abstraction model consists of fragments, and each fragment represents a single state in the abstraction model and a subset of states in the original model. The main formula is the abstraction model's algebraic expression, and the component formula is a formula related to the fragment. In this way, eFACT computes the confidence intervals for each closed-form expression, and then uses the obtained results to calculate the confidence interval for the main formula. Furthermore, eFACT exploits a binary search technique to enable engineers to obtain the highest confidence level at which a non-functional requirement of a system can be confirmed as violated or satisfied—an important feature unavailable in the FACT tool. Furthermore, since we are concerned about the safety and business-critical systems, it is vital to consider a high confidence level before deploying the system.

The paper is organised as follows. Section II explains how eFACT computes the confidence intervals for the properties of large pDTMC models. Next, Section III demonstrates the use of binary search to efficiently find the required confidence level. Section IV describes the case studies used to evaluate eFACT. Next, Section V discusses the experimental results, and Section VI compares our solution to related work. Finally, Section VII briefly summarises this work, and highlights directions for future work.

## II. CONFIDENCE INTERVALS FOR LARGE PDTMCS

For a given pDTMC model and a property of this model encoded in Probabilistic Computation Tree Logic (PCTL), FACT obtains an algebraic expression from PRISM to compute confidence intervals. However, FACT cannot produce confidence intervals when the model is large (as explained in the previous section). eFACT aims to analyse large pDTMC models with at least one unknown transition probability, provided that observations of the unknown transition exist. To achieve this purpose, we exploit a recent advance in probabilistic model checking and the model checker ePMC [4][5] that produces closed-form expressions (i.e., component formula) for the property being analysed, and then combines them into one main formula. eFACT analyses each expression separately to produce its confidence intervals for the provided confidence levels. The confidence intervals of the expression then substitute into the main formula. Therefore, the outcomes of all expressions contribute to calculating the confidence intervals for the analysed property of a given large pDTMC.

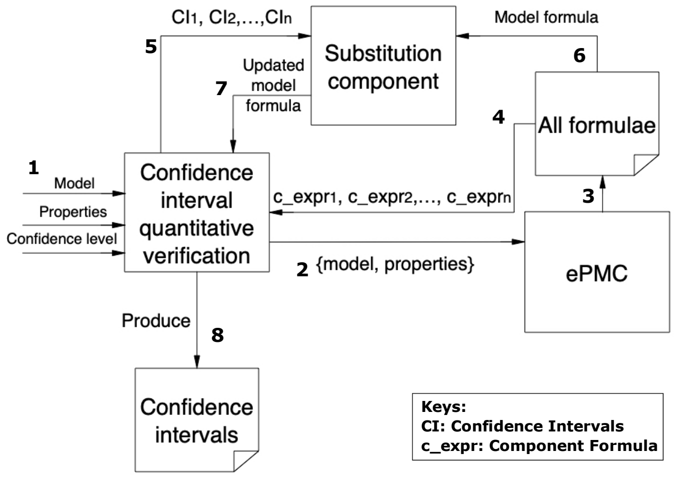


Figure 1. eFACT structure.

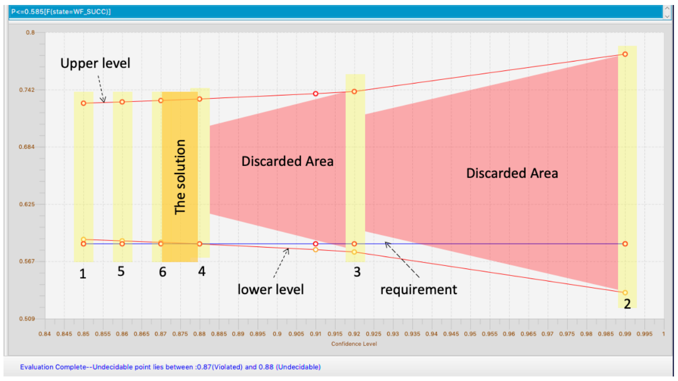
At a given confidence level $\alpha$, computing confidence intervals for a large pDTMC model consists of three main steps: confidence interval quantitative verification, ePMC, and substitution. The confidence interval quantitative verification is used to receive the inputs and compute the confidence intervals for closed-form expression. ePMC is employed to produce the closed-form expressions and the main formula of the property. The substitution component is used to substitute the outcome of all closed-form expressions in the main formula of the property. The substitution component handles two equations representing the main formula: one to substitute all lower intervals of each expression into the main formula and the other is used to substitute the upper level of expressions in the main formula.

Figure 1 illustrates the steps followed to compute the confidence intervals for a large pDTMC in detail. First, the confidence interval quantitative verification will receive the pDTMC model, property, and a range of confidence intervals as inputs. The model and property are then sent to ePMC to obtain all possible closed-form expressions formulae (Steps 2 and 3 in Figure 1). There are two kinds of produced formulae: the component formula (closed-form expression) and the model formula used to analyse the system model. The component formula could be a part of the model formula in the latter formula. In general, the formula represents an algebraic expression related to the analysed property of the model. The component formulae (denoted as $c\_expr_1$,$c\_expr_2$,...,$c\_expr_n$ in the figure) are then sent for confidence interval quantitative verification to compute their confidence intervals sequentially (Step 4). The results of analysing the component formulae are sent to the substitution component to substitute their results into the model formula (as shown in Steps 5 and 6). Finally, the model formula is sent to the confidence interval quantitative verification unit to calculate the final confidence intervals that will appear to the end-user.

## III. Finding the highest confidence level

When engineers use eFACT to compute confidence intervals for a PCTL-encoded pDTMC property, they are often interested in comparing these intervals with a bound that the property must satisfy per the analysed system's non-functional requirement. Furthermore, they are particularly interested in finding the *highest confidence level* $\alpha_{MAX}$ at which the requirement can be shown as violated or satisfied, given the available set of observations of the unknown pDTMC transitions. For confidence levels $\alpha > \alpha_{MAX}$, the observations available are insufficient for deciding whether the requirement is satisfied. Finding the value of $\alpha_{MAX}$ (or a close approximation of it) enables important decision-making. For instance, if a requirement can be shown to be satisfied at the highest confidence level $\alpha_{MAX} = 0.99$, the system can be confidently deployed (based on the requirement being met). In contrast, if a requirement can only be shown as satisfied at the highest confidence level $\alpha_{MAX} = 0.75$, the decision of whether to deploy the system cannot be made. Further observations should be obtained, for example, by testing the relevant system components.

eFACT can compute a potentially very large number of confidence intervals at different confidence levels $\alpha$ to find a close approximation of $\alpha_{MAX}$. eFACT is highly inefficient in achieving this, given the overheads of formal verification with confidence intervals. Therefore, we developed an efficient method (implemented in eFACT) for computing this close approximation. This method employs a binary search to efficiently approximate the highest confidence level $\alpha_{MAX}$. Therefore, instead of slowly performing verification for each confidence level to determine where the requirement is satisfied or violated, the binary search technique will speed up the process of achieving this. When the user inserts the model (non-functional requirement and range of confidence levels), eFACT starts its work by verifying the first inserted confidence level and computing its confidence intervals. Following this, it moves to the last inserted confidence level to calculate its confidence intervals. Now, there are two confidence levels with their intervals, enabling eFACT to check whether the analysed property requirement is located inside those intervals. If the requirement is located inside all intervals, the process will terminate with a message stating that the requirement is undecidable for the given range of confidence levels. Otherwise, eFACT moves to the middle confidence level (e.g., if the range of confidence levels is between 89 and 99, then the middle level is 94) and computes the confidence intervals. eFACT then checks the requirement's position over the current



Evaluation Complete--Undecidable point lies between :0.87(Violated) and 0.88 (Undecidable)

Figure 2. An example of using binary search in eFACT.

confidence intervals and compares it with the obtained ones over the confidence intervals from the first and last levels.

The confidence levels that lie between the middle confidence level and the other confidence level, in which the requirement's position matches its place in the middle level, will be discarded. Again, eFACT moves to the middle of the remaining confidence levels and repeats the same procedure until it determines the highest confidence level $\alpha_{MAX}$.

Figure 2 shows the verification result, where eFACT is looking for the confidence level at which the requirement is violated or satisfied. The test was conducted between confidence levels 0.85 and 0.99, where the increment step was 0.01. Instead of completing 15 verification tests to determine the required confidence level, we performed six verification tests until the required result was found. The discarded area (red area) has a list of confidence levels with intervals containing the requirement; therefore, performing additional tests in this area is useless. The solution area (green area) is where the requirement's position moves from outside the confidence intervals to be inside the next intervals.

## IV. Case Studies

### A. Service-based systems

Service-based systems (SBSs) are applications that provide services that are dependent on or connected to each other [6]. SBSs comprise internal system components and possible independent third-party components implemented as services. There are different ways in which services can conduct operations similar to those of SBSs but with different probabilities in their execution time ($t_1$,...,$t_n$), costs ($c_1$,...,$c_n$) and successes ($p_1$,...,$p_n$). The patterns are adopted from [4]: sequential execution (SEQ), sequential execution with a retry (SEQ-R), sequential execution with retry1 (SEQ-R1), probabilistic execution (PROB), probabilistic execution with a retry (PROB-R), probabilistic execution with retry1 (PROB-R1), parallel execution (PAR), and parallel execution with a retry (PAR-R). They are used to implement the SBS operations with $n$ services equivalent to those operations described below:

1) SEQ $(p_1, t_1, c_1, ..., p_n, t_n, c_n)$: There are $n$ services invoked in order, terminated after the last service or upon a first successful request.
2) SEQ-R $(p_1, t_1, c_1, ..., p_n, t_n, c_n, r)$: This is similar to SEQ. However, if all service invocations fail, the operation is re-executed from the first service with probability $r$ or it fails with probability 1-$r$.
3) SEQ-R1 $(p_1, t_1, c_1, r_1, ..., p_n, t_n, c_n, r_n)$: This is similar to SEQ. However, service $i$ will be re-invoked with probability $r_i$ if the invocation of this service fails.
4) PAR$(p_1, t_1, c_1, ..., p_n, t_n, c_n)$: There are $n$ services invoked simultaneously. The operation will use the output of the first successful invocation.
5) PAR-R $(p_1, t_1, c_1, ..., p_n, t_n, c_n, r)$: This is similar to PAR. However, if all service invocations fail, the operation is re-executed with probability $r$ or it fails with probability 1-$r$.
6) PROB $(x_1, p_1, t_1, c_1, ..., x_n, p_n, t_n, c_n)$: There is a single service to request. The probability that indicates the service $i$ is $x_i$, where $\Sigma_{i=1}^{n}$ $x_i$ =1.
7) PROB-R$(x_1, p_1, t_1, c_1, ..., x_n, p_n, t_n, c_n, r)$: This is similar to PROB. However, if the service invocations fail, the operation is re-executed with probability $r$ or it fails with probability 1-$r$.
8) PROB-R1$(x_1, p_1, t_1, c_1, r_1, ..., x_n, p_n, t_n, c_n, r_n)$: This is similar to PROB. However, if the service invocations fail, the service is re-invoked with the probability of $r$ or it fails with probability 1-$r$.
9) Combination: This is a combination of the above patterns.

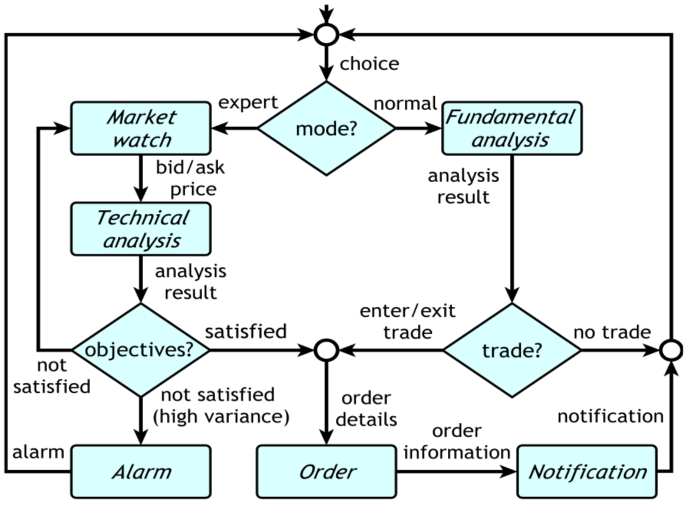


Figure 3. Foreign Exchange System Workflow, from [7].

To evaluate eFACT, a foreign exchange system (FX system) from the SBS area that aims to assists the trader is adopted from [7]. As shown in the Figure 3, the FX system offers the trader two operational modes: expert or normal. The expert mode executes the trade automatically when the transaction meets the customer's objectives. It begins with the market-watch component to obtain the current price of the chosen currency, then it uses the technical-analysis component to assess the market and estimate the price movement. The analysed outputs could be one of three options:

1) The transaction can be performed because the objectives that the traders set up are satisfied;
2) The market watch component is re-invoked since the objectives were not met; and
3) The objectives are incorrect, and the Alarm unit will be triggered to warn the trader.

Conversely, the FX system utilises the fundamental-analysis component in its normal mode to determine whether to conduct a transaction, retry the analysis or end the session.

eFACT aims to analyse the following properties of the pDTMC model of the FX system with multiple services (from 1 to 6), and under different patterns:

1) $P1$: The possibility of completing a transaction successfully, written in the PCTL format as $P=?[F(state = WF\_SUCC)]$;
2) $P2$: The estimated time to execute the transaction, written in the PCTL format as $R\{"time"\}=?[F((state = WF\_SUCC)|(state = WF\_FAIL))]$; and
3) $P3$: The estimated cost of running the transaction successfully, written in the PCTL format as $R\{"cost"\} = ?[F((state = WF\_SUCC)|(state = WF\_FAIL))]$.

## B. Three-tier software architectures

The three-tier server [8], as shown in Figure 4, provides three services: web, database and application services. The services are hosted on four different physical servers (A, B, C and D) and operate on different virtual machines (VMs). The system can be scaled-up to include more servers, VMs and service instances. This case study presents the following three patterns:

- Basic (B): Several tier instances are running on a server. If the server crashes, the running tier instances are lost.
- Virtualised (V): There are a number of tier instances, and each one is running on its own virtual machine on a server.
- Virtulised-M (VM): This is similar to the virtualised pattern. However, when the server crashes, a monitoring component can detect a crash before it occurs. Therefore, the virtual machine can be migrated to other running servers. For example, consider a server with several components, including processors, disks, and memory chips, that are now working but are prone to crash over time. If a substantial quantity of components crashes, the detection monitoring component discovers the crashes and begins migrating all the VMs in this server to another server.

If the engineers intend to evaluate the probability of deploying options for the three-tier software on different servers, they could evaluate the following properties:

1) $P1$: This measures the likelihood of the system failing within a determined time due to all tier instances failing. It can be written in PCTL as $P=?[F\ done\ \&\ fail]$; and
2) $P2$: This assesses the possibility of a single failure point during the analysis. The PCTL encoded for this property is $P=?[F\ done\ \&\ spf]$.

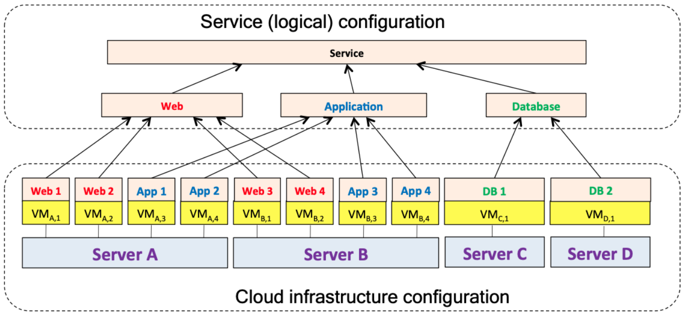


Figure 4. Three-tier architecture deployed on a Cloud, from [8].

## V. Evaluation

We performed a set of experiments to compare eFACT and FACT using two different case studies from different areas. Those case studies are described in Section IV. All experiments were conducted on an OSX 10.14.6 MacBook Pro laptop with an 8 GB 1600 MHz DDR3 RAM and CPU 2.5 GHz Intel Core i5 processor.

## A. Experimental environment

eFACT was developed using JAVA and required installing the following tools and applications:

1) PRISM/Storm [9] are model checker tools used to analyse properties and produce algebraic expressions. eFACT tested using PRISM v4.4 and Storm v1.5.1.
2) MATLAB is used for computing confidence intervals, and the used version is R2019a.
3) YALIMP [10][11] is a MATLAB-based modelling language that was developed by Johan Lofberg and contains several free and commercial solvers. It is used to model and formulate both convex and non-convex optimisation problems. It is invoked in the background by eFACT/FACT to solve the convex optimisation problem. Our work was applied using version 20210331.
4) Gurobi [12] is an optimisation solver that YALIMP can invoke to solve the optimisation problem.
5) ePMC repository defines the model's patterns and contains the expressions related to the properties of the model.

## B. Results

For the first case study that SBS explained in Section IV-A, we carried out several experiments to analyse three properties ($P1$, $P2$, $P3$), produce the confidence intervals from $\alpha$=0.90 to $\alpha = 0.99$, and record the execution time in seconds. The analysis was carried out under different patterns and with different services. Table I summarises the results and shows the execution time (in seconds) taken to analyse each property using eFACT and FACT. The table contains the following symbols:

- (T) denotes the time out, which means that the execution time exceeds the predefined time of 1800 seconds without completing the analysis.

TABLE I. THE RESULTS OF FX SYSTEM, (THE EXECUTION TIME IS IN SECONDS).

| Pattern | Services | eFACT | | | FACT | | |
|---|---|---|---|---|---|---|---|
| | | P1 | P2 | P3 | P1 | P2 | P3 |
| SEQ | 1 | 141.405 | 175.357 | 188.243 | 98.817 | 105.328 | 100.098 |
| | 2 | 147.276 | 194.089 | 194.503 | T | T | T |
| | 3 | 188.993 | 225.028 | 227.659 | - | - | - |
| | 4 | 286.416 | 354.003 | 353.514 | - | - | - |
| SEQ-R | 2 | 197.213 | 284.967 | 282.978 | T | T | T |
| | 3 | 253.189 | 348.103 | 355.679 | - | - | - |
| | 4 | 1507.257 | 1314.996 | 1285.241 | - | - | - |
| SEQ-R1 | 2 | 187.587 | 270.305 | 272.994 | T* | T | T |
| | 3 | 222.844 | 322.971 | 322.634 | - | - | - |
| | 4 | 423.71 | 501.281 | 510.492 | - | - | - |
| PAR | 2 | 141.299 | 189.637 | 185.0 | T | T | T |
| | 3 | 186.318 | 269.309 | 221.306 | - | - | - |
| | 4 | 290.874 | 384.679 | 296.634 | - | - | - |
| PAR-R | 2 | 199.079 | 299.229 | 280.785 | T | T | T |
| | 3 | 248.545 | 380.596 | 337.698 | - | - | - |
| | 4 | 1487.141 | 1787.865 | 1352.024 | - | - | - |
| PROB | 2 | 138.287 | 183.473 | 182.499 | 182.595 | 1130.434 | 1025.849 |
| | 3 | 143.724 | 187.631 | 192.325 | - | - | - |
| | 4 | 148.537 | 197.401 | 200.723 | - | - | - |
| PROB-R | 2 | 197.09 | 262.869 | 263.637 | T | T | T |
| | 3 | 220.979 | 294.346 | 295.803 | - | - | - |
| | 4 | 238.253 | 339.933 | 334.826 | - | - | - |
| PROB-R1 | 2 | 186.974 | 262.863 | 260.842 | T | T | T |
| | 3 | 216.312 | 293.574 | 294.891 | - | - | - |
| | 4 | 230.583 | 337.127 | 345.044 | - | - | - |
| Combination | Min | 155.351 | 199.017 | 197.774 | T | T | T |
| | Max | 292.297 | 290.975 | 286.386 | - | - | - |
| | Mean | 177.582 | 231.562 | 222.059 | - | - | - |
| | Stdev | 30.139 | 24.23 | 24.952 | - | - | - |

- (T*) means the tool is failed to produce an algebraic expression for the property being analysed during the predefined time.
- (-) indicates that we skipped this experiment since the previous model is smaller than the current one, and it failed to compute the confidence intervals in the determined time frame.

As shown in Table I, the execution time recorded for eFACT is better than FACT's execution time, except for the first row, where the model has a single service (SEQ pattern with one service). Further, to analyse the model in the first row, eFACT requires more time to compute confidence intervals for the component expressions (more than one expression) before substituting their results into the model formula. Moreover, we notice that the difference is not so significant. The table shows that eFACT takes less time than FACT for the analysis of other patterns and services.

For the second case study mentioned in Section IV-B, several experiments were performed to evaluate their two properties ($P1, P2$) and calculate the confidence intervals from $\alpha$=0.90 to $\alpha = 0.99$. Table II illustrates the results for four models of four servers with different patterns. FACT takes less execution time to analyse the model of deployment D1, which is found in the first row. The model is simple and produces a small expression that FACT can handle. In deployment D2, the model has some complexity (loop), and the expressions for both properties are too large. Therefore, FACT failed to analyse them before the time was out. eFACT can handle this model since it deals with small component expressions to first analyse them and then exploit their outcomes in analysing the main formula. The third row is for deployment D3, which is a loop-free model. We note that FACT can analyse this model but is higher than eFACT analysis time. The last row shows the superiority of eFACT, where FACT cannot analyse the properties of this model since the algebraic expression failed to be produced in 1800 seconds.

## VI. RELATED WORK

Software engineers can exploit the probabilistic model checking to analyse and assess the reliability, correctness, potential performance and other key attributes of systems with probabilistic behaviour. However, the model can be affected by the unquantified estimation errors of transition probabilities, leading to uncertainty. Specifically, the probabilities of transitions from one state to another in DTMC could be unrealistic since statistical experiments calculate them. Multiple studies have been conducted to diminish the uncertainty that arises in DTMC models. The studies accomplished by [13][14] have sought to capture this kind of problem. Kozine and Utkin [13] supposed that the probability value should be included between two bounds (upper and lower) instead of being a specific value. They exploited the theory of interval-valued coherent prevision to generalise discrete Markov chains and introduce interval-valued, discrete-time Markov chains (IDTMCs). Škulj [14] attempted to refine the IDTMCs and develop consecutive steps to make the IDTMCs suitable for the models with generic convex sets of probabilities. The work in [15] applied upper and lower bounds on the complexity

TABLE II. THE RESULTS OF THE MULTI-TIER SYSTEM.

| Deployment | Number of instances | Server type | | | | eFACT | | FACT | |
|---|---|---|---|---|---|---|---|---|---|
| | | Server A | Server B | Server C | Server D | P1 | P2 | P1 | P |
| D1 | 6 | V | V | B | B | 203.266 | 214.387 | 94.088 | 86.317 |
| D2 | 6 | VM | VM | B | B | 331.342 | 359.419 | T | T |
| D3 | 10 | V | V | V | V | 337.281 | 365.966 | 687.554 | 730.999 |
| D4 | 10 | VM | VM | VM | VM | 824.153 | 873.638 | T* | T* |

of calculating values for undetermined probabilities in the model checking of an interval Markov chains that increased the likelihood of satisfying $\omega$-regular specification. FACT [1] computes confidence intervals for analysing the properties of Markov chains instead of giving a single value to resolve uncertainty. However, FACT fails to compute the confidence intervals when the produced algebraic expression is too large or not produced. eFACT employs the ePMC approach to compute the confidence intervals when FACT cannot.

## VII. CONCLUSION AND FUTURE WORK

In this paper, we introduced $e$FACT, a new model checker with confidence intervals that significantly improves the scalability of existing solutions for the analysis of pDTMCs. In addition, eFACT can benefit engineers who want to establish the analysed pDTMC model's confidence level in the satisfaction or violation of a given non-functional requirement. Our experimental results show that $e$FACT has better execution times than the model checker FACT that it builds on, outperforming FACT in most cases. One of our work's limitation is that the model requires a repository of components' equations and an abstract model that require a domain expert. However, this limitation can be resolved using a recently introduced generic method for efficient parametric model checking [16]. Integrating this new method into eFACT and further evaluating the scalability of the tool represent areas of future work for our project. As another future work direction, the efficiency of eFACT can be further increased by analysing the component expressions generated by ePMC in parallel.

## REFERENCES

[1] R. Calinescu, K. Johnson, and C. Paterson, "FACT: A probabilistic model checker for formal verification with confidence intervals," in International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Springer, 2016, pp. 540–546.
[2] N. Alasmari, R. Calinescu, C. Paterson, and R. Mirandola, "Quantitative verification with adaptive uncertainty reduction," Journal of Systems and Software, 2022, in press. Pre-print available at https://www.sciencedirect.com/science/article/abs/pii/S016412122200036X.
[3] M. Kwiatkowska, G. Norman, and D. Parker, "Prism: Probabilistic symbolic model checker," in International Conference on Modelling Techniques and Tools for Computer Performance Evaluation. Springer, 2002, pp. 200–204.
[4] R. Calinescu, C. Paterson, and K. Johnson, "Efficient parametric model checking using domain knowledge," IEEE Transactions on Software Engineering, vol. 47, no. 6, pp. 1114–1133, 2019.
[5] R. Calinescu, K. Johnson, and C. Paterson, "Efficient parametric model checking using domain-specific modelling patterns," in 2018 IEEE/ACM 40th International Conference on Software Engineering: New Ideas and Emerging Technologies Results (ICSE-NIER). IEEE, 2018, pp. 61–64.
[6] M. Deubler, J. Grünbauer, J. Jürjens, and G. Wimmel, "Sound development of secure service-based systems," in Proceedings of the 2nd International Conference on Service Oriented Computing, 2004, pp. 115–124.
[7] S. Gerasimou, G. Tamburrelli, and R. Calinescu, "Search-based synthesis of probabilistic models for quality-of-service software engineering," in 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, 2015, pp. 319–330.
[8] R. Calinescu, S. Kikuchi, and K. Johnson, "Compositional reverification of probabilistic safety properties for large-scale complex IT systems," in Monterey Workshop. Springer, 2012, pp. 303–329.
[9] C. Dehnert, S. Junges, J.-P. Katoen, and M. Volk, "A storm is coming: A modern probabilistic model checker," in International Conference on Computer Aided Verification. Springer, 2017, pp. 592–600.
[10] J. Lofberg, "YALMIP : a toolbox for modeling and optimization in MATLAB," in 2004 IEEE International Conference on Robotics and Automation (IEEE Cat. No.04CH37508), 2004, pp. 284–289.
[11] J. Löfberg, "Modeling and solving uncertain optimization problems in yalmip," IFAC Proceedings Volumes, vol. 41, no. 2, pp. 1337–1341, 2008.
[12] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2021. [Online]. Available: https://www.gurobi.com
[13] I. O. Kozine and L. V. Utkin, "Interval-valued finite Markov chains," Reliable computing, vol. 8, no. 2, pp. 97–113, 2002.
[14] D. Škulj, "Discrete time Markov chains with interval probabilities," International Journal of Approximate Reasoning, vol. 50, no. 8, pp. 1314–1329, 2009.
[15] M. Benedikt, R. Lenhardt, and J. Worrell, "LTL model checking of interval Markov chains," in International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Springer, 2013, pp. 32–46.
[16] X. Fang, R. Calinescu, S. Gerasimou, and F. Alhwikem, "Fast parametric model checking through model fragmentation," in 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE). IEEE, 2021, pp. 835–846.