Efficient Parametric Model Checking Using Domain Knowledge

Radu Calinescu, Colin Paterson, and Kenneth Johnson

Abstract—We introduce an efficient parametric model checking (ePMC) method for the analysis of reliability, performance and other quality-of-service (QoS) properties of software systems. ePMC speeds up the analysis of parametric Markov chains modelling the behaviour of software by exploiting domain-specific modelling patterns for the software components (e.g., patterns modelling the invocation of functionally-equivalent services used to jointly implement the same operation within service-based systems, or the deployment of the components of multi-tier software systems across multiple servers). To this end, ePMC precomputes closed-form expressions for key QoS properties of such patterns, and uses these expressions in the analysis of whole-system models. To evaluate ePMC, we show that its application to service-based systems and multi-tier software architectures reduces the analysis time by several orders of magnitude compared to current parametric model checking methods.

Index Terms—Parametric model checking; Markov models; model abstraction; probabilistic model checking; quality of service

1 Introduction

Parametric model checking (PMC) [19], [32], [34] is a formal technique for the analysis of Markov chains with transition probabilities specified as rational functions over a set of continuous variables. When the analysed Markov chains model software systems, these variables represent configurable parameters of the software or environment parameters unknown until runtime. The properties of Markov chains analysed by PMC are formally expressed in probabilistic computation tree logic (PCTL) [33] extended with rewards [2], and the results of the analysis are algebraic expressions over the same variables.

In software engineering, Markov chains are used to model the stochastic nature of software aspects including user inputs, execution paths and component failures, and the expressions generated by PMC correspond to reliability, performance and other quality-of-service (QoS) properties of the analysed software. The availability of algebraic expressions for these key QoS properties has multiple applications. First, evaluating the expressions for different parameter values enables the fast comparison of alternative system designs, e.g., in software product lines [28], [29]. Second, self-adaptive software can efficiently evaluate the expressions at runtime, when the unknown environment parameters can be measured and suitable new values for the configuration parameters need to be selected [22]. Third, PMC expressions allow the algebraic calculation of parameter values such that a QoS property satisfies a given constraint [19]. Finally, they enable the precise analysis of the sensitivity of QoS properties to changes in the system parameters [24].

PMC is supported by the model checkers PARAM [31], PRISM [38] and Storm [21]. However, despite significant advances in recent years [19], [32], [34], the current PMC techniques (which these model checkers implement) are computationally very expensive, generate expressions that are often extremely large and inefficient to evaluate, and do not support the analysis of parametric Markov chains modelling important classes of software systems.

Our work addresses these major limitations of existing PMC techniques and tools. To this end, we introduce an efficient parametric model checking (ePMC) method that exploits *domain-specific modelling patterns*, i.e., "fragments" of parametric Markov chains occurring frequently in models of software systems from a domain of interest, and corresponding to typical ways of architecting software components within that domain.

As shown in Fig. 1, ePMC comprises two stages. The first stage is performed only once for each domain that ePMC is applied to. This stage uses domain-expert input to identify modelling patterns for components of systems from the considered domain, and precomputes closed-form expressions for key QoS properties of these patterns. For example, the modelling patterns for the service-based systems domain (described in detail in Section 6) correspond to different ways in which $n \geq 1$ functionally-equivalent services can be used to execute an operation of the system. One option is to invoke the n services *sequentially*, such that service 1 is always invoked, and service $i > 1$ is only invoked if the invocations of services $1, 2, \ldots, i{-}1$ have all failed. The component modelling pattern labelled 'SEQ' at the top of Fig. 1 depicts this option. The graphical representation of the pattern shows the invocations of the n services as states labelled $1, 2, \ldots, n$, and the successful and failed completion of the operation as states labelled with a tick '✓' and a cross '✗', respectively. QoS properties such as the probability of reaching the success state and the expected execution time and cost of the operation for this pattern can be computed as

$$prob = p_1 + (1-p_1)p_2 + \ldots + \left(\prod_{i=1}^{n-1}(1-p_i)\right)p_n$$
$$time = t_1 + (1-p_1)t_2 + \ldots + \left(\prod_{i=1}^{n-1}(1-p_i)\right)t_n$$

- *R. Calinescu and C. Paterson are with the Department of Computer Science at the University of York, UK.*
- *K. Johnson is with the School of Engineering, Computer and Mathematical Sciences at the Auckland University of Technology, New Zealand.*

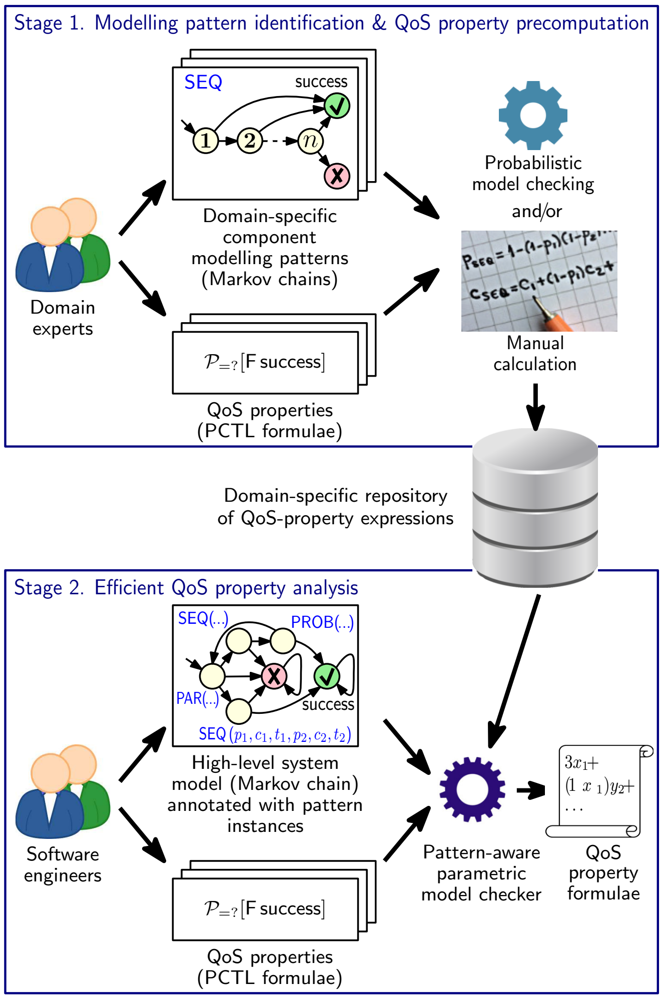

Stage 1. Modelling pattern identification & QoS property precomputation
SEQ
success
Domain experts
Domain-specific component modelling patterns (Markov chains)
$\mathcal{P}_{=?}$[F success]
QoS properties (PCTL formulae)
Probabilistic model checking
and/or
Manual calculation
Domain-specific repository of QoS-property expressions
Stage 2. Efficient QoS property analysis
SEQ(...)
PROB(...)
PAR(...)
success
SEQ $(p_1, c_1, t_1, p_2, c_2, t_2)$
Software engineers
High-level system model (Markov chain) annotated with pattern instances
$\mathcal{P}_{=?}$[F success]
QoS properties (PCTL formulae)
Pattern-aware parametric model checker
$3x_1+$ $(1\ x\ _1)y_2+$ ...
QoS property formulae

Fig. 1. Two-stage efficient parametric model checking

$$cost = c_1 + (1-p_1)c_2 + \ldots + \left(\prod_{i=1}^{n-1}(1-p_i)\right)c_n$$

where p_i, t_i and c_i are the probability of successful invocation, the execution time and the cost of service i, $1 \leq i \leq n$, respectively. As illustrated in Fig. 1, these calculations can be carried out using an existing probabilistic model checker or manually. The resulting expressions are stored in a domain-specific repository, and are used in the next ePMC stage.

The second ePMC stage is performed for each structurally different variant of a system and QoS property under analysis. The stage involves the PMC of a parametric Markov chain that models the interactions between the system components. This Markov model can be provided by software engineers with PMC expertise, or can be generated from more general software models, such as UML activity diagrams annotated with probabilities as in [7], [18], [25]. The model states associated with system components are labelled with *pattern instances* that specify the modelling pattern used for each component and its parameters. For instance, the pattern instance SEQ$(p_1, c_1, t_1, p_2, c_2, t_2)$ from Fig. 1 labels a component implemented using the sequential pattern described earlier and $n = 2$ services with success probabilities p_1, p_2, costs c_1, c_2 and mean execution times t_1, t_2. The pattern-annotated Markov model is analysed by a model checker with pattern manipulation capabilities. The result of the analysis is a set of formulae comprising:

- A formula for the system-level QoS property, specified as a function over the component-level QoS property values. This formula is obtained by applying standard PMC to the pattern-annotated Markov model;
- Formulae for the relevant component-level QoS properties. These formulae are obtained by instantiating the appropriate closed-form expressions from the domain-specific repository produced in the first ePMC stage.

All ePMC formulae are rational functions that can be efficiently evaluated for any combinations of parameter values, e.g., using tools such as Matlab and Octave.

The main contributions of our paper are:

1) A theoretical foundation for the ePMC method.
2) An open-source tool that automates the application of the method, and is freely available from our project website https://www.cs.york.ac.uk/tasp/ePMC/.
3) Repositories of modelling patterns for the service-based systems and multi-tier software architecture domains.
4) An extensive evaluation which shows that ePMC is several orders of magnitude faster and produces much smaller algebraic expressions compared to the PMC techniques currently implemented by the leading model checkers PARAM, PRISM and Storm, in addition to supporting the analysis of parametric Markov chains that are too large for these model checkers.

These contributions build on our preliminary work from [12], extending it with a theoretical foundation, tool support, repositories of modelling patterns for two domains, and a significantly larger evaluation.

The rest of the paper is structured as follows. Section 2 provides a brief introduction to the model checking of parametric Markov chains. Section 3 describes a simple service-based system that we then use as a running example when presenting the ePMC theoretical foundation in Section 4. Section 5 covers the implementation of the ePMC tool, while Sections 6 and 7 detail the application of ePMC to the service-based systems and multi-tier software architectures domains, respectively. Section 8 presents our experimental results, and Section 9 compares our method with related work. Finally, Section 10 provides a brief summary and discusses our plans for future work.

2 PRELIMINARIES

2.1 Parametric Markov chains

Markov chains (MCs) are finite state transition systems used to model the stochastic behaviour of real-world systems. MC states correspond to relevant configurations of the modelled system, and are labelled with atomic propositions which hold in those states. State transitions model all possible transitions between states, and are annotated with probabilities as specified by the following definition.

Definition 1. A Markov chain M over a set of atomic propositions AP is a tuple

$$M = (S, s_0, \mathbf{P}, L), \tag{1}$$

where S is the finite set of MC states; $s_0 \in S$ is the initial state; $\mathbf{P} : S \times S \to [0,1]$ is a transition probability matrix where, for any states $s, s' \in S$, $\mathbf{P}(s, s')$ is the probability of transitioning to state s from state s'; and $L : S \to 2^{AP}$ is the state labelling function.

A state s of a Markov chain M is an *absorbing state* if $\mathbf{P}(s, s) = 1$ and $\mathbf{P}(s, s') = 0$ for all $s' \neq s$, and a *transient state* otherwise. A *path* π over M is a possibly infinite sequence of states from S such that for any adjacent states s and s' in π, $\mathbf{P}(s, s') > 0$. The m-th state on a path π, $m \geq 1$, is denoted $\pi(m)$. For any state s, $Paths^M(s)$ represents the set of all infinite paths over M that start with state s. Finally, we assume that every state $s \in S$ is reachable from the initial state, i.e., there exists a path $\pi \in Paths^M(s_0)$ such that $\pi(i) = s$ for some $i > 0$.

To compute the probability that a Markov chain (1) behaves in a specified way when in state s, we use a *probability measure* $\Pr_s$ defined over $Paths^M(s)$ such that [3], [37]:

$$\Pr_s(\{\pi \in Paths^M(s) \,|\, \pi = s_1 s_2 \ldots s_m \ldots\}) = \mathbf{P}(s_1 s_2 \ldots s_m) = \textstyle\prod_{i=1}^{m-1} \mathbf{P}(s_i, s_{i+1}),$$

where $\{\pi \in Paths^M \mid \pi = s_1 s_2 \ldots s_m \ldots\}$ is the set of all infinite paths that start with the prefix $s_1 s_2 \ldots s_m$ (i.e., the *cylinder set* of this prefix). Further details about this probability measure and its properties are available from [3], [37].

To allow the verification of a broader set of QoS properties, MC states can be annotated with nonnegative values termed *rewards* [2]. These values are interpreted as "costs" (e.g. energy used) or "gains" (e.g. requests processed).

Definition 2. A *reward structure* over a Markov chain $M = (S, s_0, \mathbf{P}, L)$ is a function $\rho : S \to \mathbb{R}_{\geq 0}$. For any state $s \in S$, $\rho(s)$ represents the reward "earned" on leaving state s.

Our work focuses on the analysis of parametric Markov chains (sometimes called *incomplete Markov chains* [5] or *uncertain Markov chains* [40]).

Definition 3. A *parametric Markov chain* is an MC comprising transition probabilities $\mathbf{P}(s, s')$ and/or rewards $\rho(s)$ defined as rational functions over a set of continuous variables [19], [32], [34].

2.2 Property specification

The properties of Markov chains are formally expressed in probabilistic variants of temporal logic. In our work we use probabilistic computation tree logic (PCTL) [17], [33] extended with rewards [2], which is supported by all leading probabilistic model checkers. Rewards-extended PCTL allows the specification of probabilistic and reward properties using the probabilistic operator $\mathcal{P}_{\bowtie p}[\cdot]$ and the reward operator $\mathcal{R}_{\bowtie r}[\cdot]$, respectively, where $p \in [0, 1]$ is a probability bound, $r \in \mathbb{R}_{\geq 0}$ is a reward bound, and $\bowtie \in \{\geq, >, <, \leq\}$ is a relational operator. Formally, a *state formula* Φ and a *path formula* Ψ in PCTL are defined by the grammar:

$$\Phi ::= true \mid a \mid \Phi \wedge \Phi \mid \neg\Phi \mid \mathcal{P}_{\bowtie p}[\Psi] \tag{2}$$

$$\Psi ::= X\Phi \mid \Phi \ \mathrm{U}\ \Phi \mid \Phi \ \mathrm{U}^{\leq k}\ \Phi \tag{3}$$

and a *reward state formula* is defined by the grammar:

$$\Phi ::= \mathcal{R}_{\bowtie r}[\mathrm{I}^{=k}] \mid \mathcal{R}_{\bowtie r}[\mathrm{C}^{\leq k}] \mid \mathcal{R}_{\bowtie r}[\mathrm{F}\ \Phi] \mid \mathcal{R}_{\bowtie r}[\mathrm{S}], \tag{4}$$

where $k \in \mathbb{N}_{>0}$ is a timestep bound and $a \in AP$ is an atomic proposition. When multiple reward structures are defined over a Markov chain, the extended notation $\mathcal{R}^{\mathsf{rwd}}_{\bowtie r}[\mathrm{I}^{=k}]$, $\mathcal{R}^{\mathsf{rwd}}_{\bowtie r}[\mathrm{C}^{\leq k}]$, etc. is used to specify that a reward state formula refers to the reward structure named 'rwd'.

The PCTL semantics is defined using a satisfaction relation $\models$ over the states S and the paths $Paths^M(s)$, $s \in S$, of a Markov chain (1). Given a state s and a path π of the Markov chain, $s \models \Phi$ means "Φ holds in state s", $\pi \models \Psi$ means "Ψ holds for path π", and we have:

– $s \models true$ for all $s \in S$;
– $s \models a$ 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 $s \models \Phi_2$;
– $s \models \mathcal{P}_{\bowtie p}[\Psi]$ iff $\Pr_s(\{\pi \in Paths^M(s) \mid \pi \models \Psi\}) \bowtie p$.
– the *next path formula* $X\Phi$ holds for path π iff $\pi(2) \models \Phi$;
– the *time-bounded until path formula* $\Phi_1 \ \mathrm{U}^{\leq k}\ \Phi_2$ holds for path π iff Φ_2 holds in the i-th path state, $i \leq k$, and Φ_1 holds in the first $i - 1$ path states, i.e.:

$$\exists i \leq k.(\pi(i) \models \Phi_2 \wedge \forall j < i.\pi(j) \models \Phi_1).$$

– the *unbounded until formula* $\Phi_1 \ \mathrm{U}\ \Phi_2$ removes the bound k from the time-bounded "until" formula.

The notation $\mathrm{F}\ \Phi \equiv true\ \mathrm{U}\ \Phi$ is used when the first part of an until formula is *true*. Thus, the *reachability property* $\mathcal{P}_{\bowtie p}[\mathrm{F}\ \Phi]$ holds if the probability of reaching a state where Φ is true satisfies $\bowtie p$. Finally, the reward state formulae specify the expected values for: the *instantaneous reward* at timestep k, $\mathcal{R}_{\bowtie r}[\mathrm{I}^{=k}]$; the *cumulative reward* up to timestep k, $\mathcal{R}_{\bowtie r}[\mathrm{C}^{\leq k}]$; the *reachability reward* cumulated until reaching a state that satisfies a property Φ, $\mathcal{R}_{\bowtie r}[\mathrm{F}\ \Phi]$; and the *steady-state reward* in the long run, $\mathcal{R}_{\bowtie r}[\mathrm{S}]$. For a detailed description of the PCTL semantics, see [2], [17], [33].

2.3 Parametric model checking

Probabilistic model checkers including MRMC [36], PRISM [38] and Storm [21] support the verification of PCTL properties of Markov chains. To verify whether a formula $\mathcal{P}_{\bowtie p}[\Psi]$ holds in a state s, these tools first compute the probability p' that Ψ holds for MC paths starting at s, and then compare p' to the bound p. The actual probability p' can also be returned (for the outermost operator $\mathcal{P}$ of a formula) so PCTL was extended to include the formula $\mathcal{P}_{=?}[\Psi]$ denoting this probability. Likewise, the extended-PCTL formulae $\mathcal{R}_{=?}[\mathrm{I}^{=k}]$, $\mathcal{R}_{=?}[\mathrm{C}^{\leq k}]$, $\mathcal{R}_{=?}[\mathrm{F}\ \Phi]$, and $\mathcal{R}_{=?}[\mathrm{S}]$ denote the actual values of the expected rewards from (4).

Parametric model checking (PMC) represents the verification of *quantitative PCTL properties* $\mathcal{P}_{=?}[.]$ without nested probabilistic operators and reward properties $\mathcal{R}_{=?}[.]$ of parametric Markov chains using algorithms such as [19], [32], [34]. The PMC verification result is a rational function of the variables used to define the transition probabilities of the verified parametric Markov chain. PMC is supported by verification tools including the dedicated model checker PARAM [31], the latest versions of PRISM [38], and the recently released model checker Storm [21].

3 Running Example

We will illustrate the theoretical aspects and the application of our ePMC method using a service-based system that

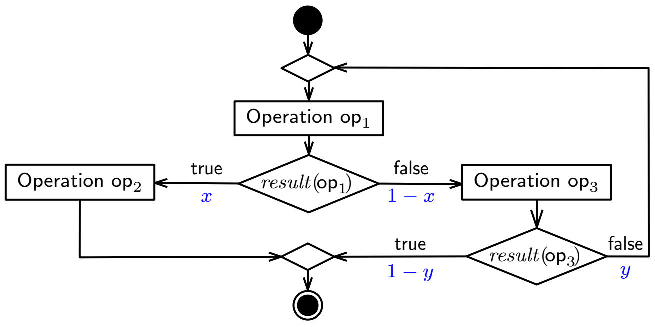


Fig. 2. UML activity diagram of the system used as a running example

implements the simple workflow from Fig. 2. This workflow handles user requests by first performing an operation op$_1$. Depending on the result of op$_1$, its execution is followed by the execution of either operation op$_2$ or operation op$_3$. The execution of op$_2$ completes the workflow, while after the execution of op$_3$ the workflow may terminate or may need to re-execute op$_1$. The outgoing branches of the decision nodes from Fig. 2 are annotated with their unknown probabilities of execution (x and $1-x$, and y and $1-y$).

We suppose that multiple functionally-equivalent services svc$_{i1}$, svc$_{i2}$, … can be used to perform each operation op$_i$, $i \in \{1, 2, 3\}$, and that these services have probabilities of successful invocation $p_{i1}, p_{i2}, \ldots$, expected response times $t_{i1}, t_{i2}, \ldots$ and invocation costs $c_{i1}, c_{i2}, \ldots$ Accordingly, the workflow can be implemented using different system architectures and service combinations. Our running example considers the implementation where:

- Operation op$_1$ is executed by invoking services svc$_{11}$ and svc$_{12}$ *sequentially*, such that service svc$_{11}$ is always invoked, and service svc$_{12}$ is only invoked if the invocation of svc$_{11}$ fails (i.e., times out or returns an error). As a result, the operation completes successfully whenever either service invocation is successful, and fails when the invocations of both services fail.
- Operation op$_2$ is executed using services svc$_{21}$ and svc$_{22}$ *probabilistically*, such that svc$_{2j}$ is invoked with probability α_j for $j \in \{1, 2\}$, where $\alpha_1 + \alpha_2 = 1$.
- Operation op$_3$ is executed by invoking services svc$_{31}$ and svc$_{32}$ *sequentially with retry*. This involves invoking the two services sequentially (as for op$_1$) and, if both service invocations fail, retrying the execution of the operation by using the same strategy with probability $r \in (0, 1)$.

The parametric Markov chain from Fig. 3a models this implementation of the workflow. For instance, the MC states s_0 and s_1 (labelled 'op$_1$') model the execution of operation op$_1$ by first invoking service svc$_{11}$ (state s_0) and, if service service svc$_{11}$ fails (which happens with probability $1 - p_{11}$), also invoking svc$_{12}$ (state s_1). The invocation of svc$_{12}$ fails with probability $1-p_{12}$, in which case the system transitions to state s_3 and then to the 'fail' state s_{13}. If either svc$_{11}$ or svc$_{12}$ succeeds (state s_2), there is a probability x that operation op$_2$ (modelled by states s_4–s_8) is executed next, and a probability $1 - x$ that the next operation is op$_3$ (modelled by states s_9–s_{12}).

To model the execution of operation op$_2$, the MC includes transitions with probabilities α_1 and α_2 from s_4 to state s_5 (which corresponds to the invocation of service svc$_{21}$) and to state s_6 (which corresponds to the invocation of service svc$_{22}$), respectively. The successful execution of svc$_{21}$ or svc$_{22}$ results in state s_7 being reached and in the successful completion of the workflow (state s_{14}, labelled 'succ'), while a failed invocation results in state s_8 being reached and in the failure of the workflow (state s_{13}). States s_9–s_{12} model the execution of operation op$_3$ similarly to how op$_1$ is modelled by s_0–s_3, except that a successful execution of op$_3$ is followed by op$_1$ (with probability y) or the successful end of the workflow (with probability $1-y$), and failed invocations of svc$_{32}$ lead to a retry of the operation (with probability r) or to the failure of op$_3$ and thus of the entire workflow (with probability $1-r$). Finally, two reward structures are defined in Fig. 3a. These structures, named 'time' and 'cost', map the MC states associated with service invocations to the expected execution time and the cost of these services, respectively.

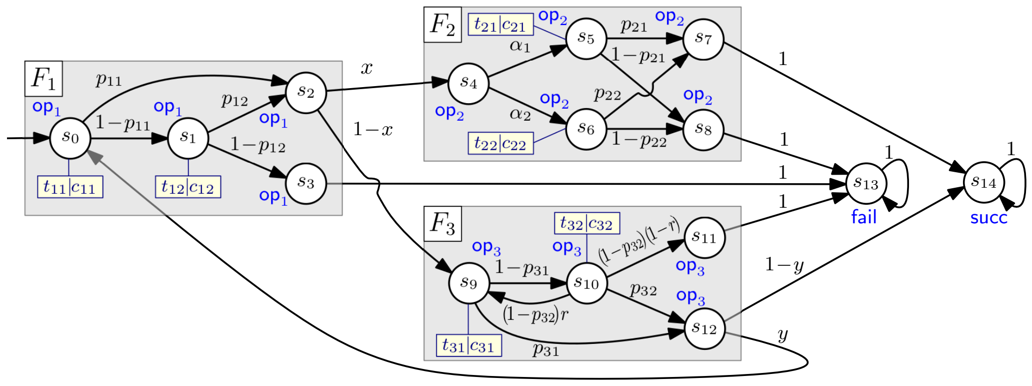
Parametric model checking applied to the MC from Fig. 3a can compute closed-form expressions for a wide range of QoS properties of the system. These properties can then be evaluated very efficiently for different combinations of services with different parameters. For our running example, we assume that the software engineers developing the system are interested to analyse the following properties:

(i) The probability P_1 that the workflow implemented by the system completes successfully;
(ii) The probability P_2 that the workflow fails due to a failed execution of operations op$_1$ or op$_2$;
(iii) The expected execution time T of the workflow;
(iv) The expected cost C of executing the workflow.

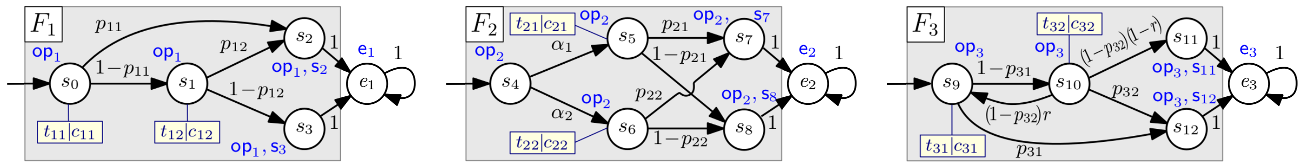
Table 1 presents these properties formalised in PCTL, and their closed-form expressions computed using the probabilistic model checker Storm and significantly simplified through manual factorisation. These results, included in order to show how large and complex the PMC expressions can be even for a simple model, already suggest that PMC might not be feasible for larger systems and models. The experimental results presented later in Section 8 confirm that indeed the PMC techniques implemented by current model checkers do not scale to much larger systems than the one from Fig. 2—a limitation addressed by our ePMC method described next.

4 ePMC Theoretical Foundation

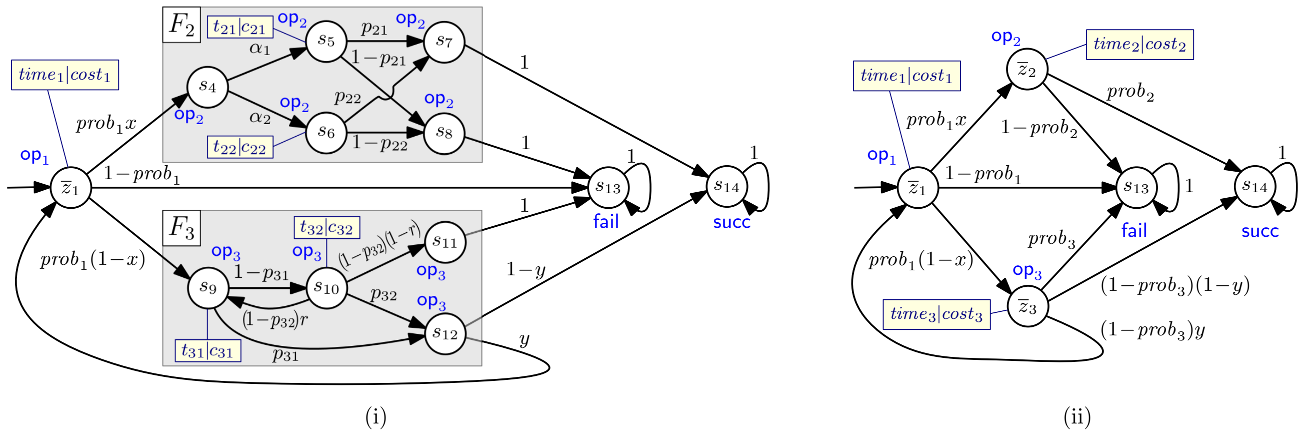
ePMC is essentially a model abstraction method, as advocated, among others, by Abrial [1] and by McIver and Morgan [39]. Given a parametric MC that is too large to be efficiently analysed or whose analysis is unfeasible, ePMC derives a smaller, abstract version of it. The ePMC abstraction method may be applicable repeatedly, yielding successively smaller abstract models, potentially leading to an abstract model that is sufficiently small to be analysed efficiently. When this analysis is feasible, its results for certain types of PCTL properties are equivalent to those that would have been obtained by analysing the initial model. The method works by replacing a *fragment* of the original model (i.e., a subset of states that satisfy specific constraints) with a single state, thus generating an *abstract MC induced by the fragment*. Furthermore, closed-form expressions for the



a. Parametric Markov chain model of the workflow implementation under analysis, with shaded fragments F_1 (corresponding to operation op$_1$), F_2 (corresponding to op$_2$) and F_3 (corresponding to op$_3$). The '$t_{ij}|c_{ij}$' combinations annotating the states that model service invocations define 'time' and 'cost' reward structures for analysing the expected execution *time* and *cost* of the workflow, respectively.



b. Parametric Markov chains associated with fragments F_1–F_3 from Fig. 3a



c. Abstract Markov chains induced by (i) fragment F_1, and (ii) fragments F_1–F_3 of the parametric Markov chain from Fig. 3a

Fig. 3. ePMC application to a parametric Markov chain model of the workflow from the running example

TABLE 1
Parametric model checking of the four QoS properties from the running example

Prop.	PCTL formula	PMC expression
P_1	$\mathcal{P}_{=?}[\mathsf{F}\ \mathsf{succ}]$	$\{p_{11}p_{12}\big[x(\alpha_2 p_{31}p_{22}p_{32}r - \alpha_1 p_{31}p_{21}r + \alpha_1 p_{21}r - \alpha_1 p_{21}p_{32}r - \alpha_2 p_{22} + \alpha_1 p_{31}p_{21}p_{32}r - yp_{32} + p_{31} - p_{31}p_{32} + p_{31}yp_{32} - \alpha_2 p_{22}p_{32}r + \alpha_2 p_{22}r + p_{32} - p_{31}y - \alpha_2 p_{31}p_{22}r - \alpha_1 p_{21})\big] + p_{11}\big[x(\alpha_1(p_{21} - p_{31}p_{21}p_{32}r + p_{21}p_{32}r - p_{21}r + p_{31}p_{21}r) + \alpha_2(p_{31}p_{22}r - p_{31}p_{22}p_{32}r - p_{22}r + p_{22}p_{32}r + p_{22}))\big] + p_{12}\big[x(\alpha_1(p_{21}p_{32}r - p_{21}r - p_{31}p_{21}p_{32}r + p_{21} + p_{31}p_{21}r) + \alpha_2(p_{22} + p_{22}p_{32}r - p_{22}r + p_{31}p_{22}r - p_{31}p_{22}p_{32}r)) + (p_{11}p_{12} - (p_{11} + p_{12})(1-x))(p_{31}p_{32} - p_{31} - p_{32})(1-y)\big]\}/\{\big[(1-x)y(p_{11} + p_{12} - p_{12}p_{12}) - r\big](p_{31}p_{32} - p_{31} - p_{32}) + 1 - r\}$
P_2	$\mathcal{P}_{=?}[\neg\mathsf{op}_3\ \mathrm{U}\ \mathsf{fail}]$	$p_{11}\big[x(\alpha_1(p_{12}p_{21} - p_{12} + 1 - p_{21}) + \alpha_2(p_{12}p_{22} - p_{12} + 1 - p_{22})) + p_{12} - 1\big] + p_{12}\big[x(\alpha_1(1-p_{21}) + \alpha_2(1-p_{22})) - 1\big] + 1$
T	$\mathcal{R}^{\mathsf{time}}_{=?}[\mathsf{F}\ \mathsf{succ} \vee \mathsf{fail}]$	$\{p_{11}p_{12}\big[x(\alpha_1(t_{21}r - t_{21} - t_{21}p_{32}r - p_{31}t_{21}r + p_{31}t_{21}p_{32}r) + \alpha_2(p_{31}t_{22}p_{32}r - t_{22} + t_{22}r - t_{22}p_{32}r - p_{31}t_{22}r) + t_{32} + t_{31} - p_{31}t_{32}) + t_{12}p_{31}p_{32}r - t_{32} - t_{31} + p_{31}t_{32} - t_{12} + t_{12}r - t_{12}p_{32}r - t_{12}p_{31}r\big] + p_{11}\big[x(\alpha_2(t_{22} - t_{22}r + t_{22}p_{32}r + p_{31}t_{22}r - p_{31}t_{22}p_{32}r) + \alpha_1(t_{21} - t_{21}r + t_{21}p_{32}r + p_{31}t_{21}r - p_{31}t_{21}p_{32}r) - t_{32} - t_{31} + p_{31}t_{32}) + t_{32} + t_{31} - p_{31}t_{32} - t_{12}r + t_{12}p_{32}r + t_{12}p_{31}r - t_{12}p_{31}p_{32}r - t_{11} + t_{11}r - t_{11}p_{32}r - t_{11}p_{31}r + t_{11}p_{31}p_{32}r + t_{12}\big] + p_{12}\big[x(\alpha_1(t_{21}p_{32}r + p_{31}t_{21}r - p_{31}t_{21}p_{32}r + t_{21} - t_{21}r) + \alpha_2(t_{22} - t_{22}r + t_{22}p_{32}r + p_{31}t_{22}r - p_{31}t_{22}p_{32}r) - t_{32} - t_{31} + p_{31}t_{32}) + t_{31} - p_{31}t_{32} + t_{32} + t_{12} - t_{12}r + t_{12}p_{32}r + t_{12}p_{31}r - t_{12}p_{31}p_{32}r\big] + t_{11} - t_{11}r + t_{11}p_{32}r + t_{11}p_{31}r - t_{11}p_{31}p_{32}r\}/\{\big[(1-x)y(p_{11} + p_{12} - p_{11}p_{12}) - r\big](p_{31}p_{32} - p_{31} - p_{32}) + 1 - r\}$
C	$\mathcal{R}^{\mathsf{cost}}_{=?}[\mathsf{F}\ \mathsf{succ} \vee \mathsf{fail}]$	expression (similar to the PMC expression for property T) not included for brevity, but available on project website

outgoing transition probabilities and rewards of the new state can be obtained through the analysis of a *parametric MC associated with the fragment*, where the size of this MC is similar to that of the fragment. We formally define these concepts below.

Definition 4. A *fragment* of a parametric Markov chain $M = (S, s_0, \mathbf{P}, L)$ is a tuple $F = (Z, z_0, Z_{\mathsf{out}})$, where:

- $Z \subset S$ is a subset of transient MC states;
- z_0 is the (only) *entry state* of F, i.e., $\{z_0\} = \{z \in Z \mid \exists s \in S \setminus Z \,.\, \mathbf{P}(s, z) > 0\}$;
- $Z_{\mathsf{out}} = \{z \in Z \mid \exists s \in S \setminus Z \,.\, \mathbf{P}(z, s) > 0\}$ is the non-empty set of *output states* of F, and all outgoing transitions from the output states are to states outside Z, i.e., $\mathbf{P}(z, z') = 0$ for all $(z, z') \in Z_{\mathsf{out}} \times Z$.

According to this definition, the output states of a fragment cannot have outgoing transitions to other states within the same fragment. However, a simple model transformation can be used to form a fragment from a state set Z that includes states z with outgoing transitions to states both inside and outside Z. This transformation replaces each such state z with states z' and z'' such that: z' "inherits" all incoming transitions of z and its outgoing transitions to states within Z, and has an additional outgoing transition to z'' (with transition probability calculated so that the transition probabilities of z' add up to 1.0); and z'' only has the incoming transition from z', and inherits the output transitions from z to states outside Z (with transition probabilities scaled to add up to 1.0).

Example 1. The shaded areas of the parametric MC from Fig. 3a (each corresponding to an operation of the workflow from our running example) contain three MC fragments:

$$\begin{aligned} F_1 &= (\{s_0, s_1, s_2, s_3\}, s_0, \{s_2, s_3\}) \\ F_2 &= (\{s_4, s_5, s_6, s_7, s_8\}, s_4, \{s_7, s_8\}) \\ F_3 &= (\{s_9, s_{10}, s_{11}, s_{12}\}, s_9, \{s_{11}, s_{12}\}) \end{aligned}$$

As shown by this example, MC fragments may or may not contain cycles.

Given a fragment F of a parametric MC M, ePMC performs parametric model checking by separately analysing two parametric MCs determined by F, and combining the results of the two analyses. As each of the two parametric MCs has fewer states and transitions than M, the overall result can be obtained in a fraction of the time required to analyse the original model M. The first of these parametric MCs is defined below.

Definition 5. The *Markov chain associated with a fragment* $F = (Z, z_0, Z_{\mathsf{out}})$ of a parametric MC $M = (S, s_0, \mathbf{P}, L)$ is the Markov chain $M_Z = (Z \cup \{e\}, z_0, \mathbf{P}_Z, L_Z)$, where e is an additional, "end" state, the transition probability matrix $\mathbf{P}_Z : (Z \cup \{e\}) \times (Z \cup \{e\}) \to [0, 1]$ is given by

$$\mathbf{P}_Z(z, z') = \begin{cases} 1, & \text{if } z \in Z_{\mathsf{out}} \cup \{e\} \wedge z' = e \\ \mathbf{P}(z, z'), & \text{if } z \in Z \setminus Z_{\mathsf{out}} \wedge z' \neq e \\ 0, & \text{otherwise} \end{cases},$$

and the atomic propositions for state $z \in Z$ are given by

$$L_Z(z) = \begin{cases} L(z) \cup \{\mathsf{z}\}, & \text{if } z \in Z_{\mathsf{out}} \\ \{\mathsf{e}\}, & \text{if } z = e \\ L(z), & \text{otherwise} \end{cases},$$

where z and e are atomic propositions that hold in state $z \in Z_{\mathsf{out}}$ and state e, respectively.[1] Additionally, any reward structure $\rho : S \to \mathbb{R}_{\geq 0}$ over M naturally maps to a reward structure $\rho_Z : Z \cup \{e\} \to \mathbb{R}_{\geq 0}$ over M_Z, where $\rho_Z(e) = 0$ and, for all $z \in Z$, $\rho_Z(z) = \rho(z)$.

Example 2. Fig. 3b shows the parametric MCs associated with fragments F_1–F_3 from Example 1, obtained by:

(i) adding transitions of probability 1 from the output states in $Z_{\mathsf{out}_1} = \{s_2, s_3\}$, $Z_{\mathsf{out}_2} = \{s_7, s_8\}$ and $Z_{\mathsf{out}_3} = \{s_{11}, s_{12}\}$ to additional states e_1, e_2 and e_3, respectively;

(ii) labelling the output states with the additional atomic propositions s_2 and s_3, s_7 and s_8, and s_{11} and s_{12}, and the end states with the new atomic propositions e_1 to e_3.

The second parametric MC determined by a fragment F and analysed by ePMC is obtained from the original MC by replacing all states from F with a single state.

Definition 6. Given a fragment $F = (Z, z_0, Z_{\mathsf{out}})$ of a parametric Markov chain $M = (S, s_0, \mathbf{P}, L)$, the *abstract MC induced by* F is $M' = (S', s'_0, \mathbf{P}', L')$, where:

- The state set $S' = (S \setminus Z) \cup \{\overline{z}\}$, where $\overline{z}$ is a new, *abstract state* that stands for all the states from Z;
- The initial state $s'_0 = s_0$, if s_0 is not the initial state of Z (i.e., $z_0 \neq s_0$), and $s'_0 = \overline{z}$ otherwise;
- The transition probability between states $s, s' \in S'$ is

$$\mathbf{P}'(s, s') = \begin{cases} \mathbf{P}(s, s'), & \text{if } s \neq \overline{z} \wedge s' \neq \overline{z} \\ \mathbf{P}(s, z_0), & \text{if } s \neq \overline{z} \wedge s' = \overline{z} \\ \sum\limits_{z \in Z_{\mathsf{out}}} prob_z \cdot \mathbf{P}(z, s'), & \text{if } s = \overline{z} \wedge s' \neq \overline{z} \\ 0, & \text{otherwise} \end{cases}$$

where

$$prob_z = \mathcal{P}_{=?}[\mathrm{F}\ \mathsf{z}] \tag{5}$$

is a reachability property calculated over the parametric Markov chain associated with the fragment F, for all output states $z \in Z_{\mathsf{out}}$;[2]

- The labelling function L' coincides with L for the states from the original MC, and maps the new state $\overline{z}$ to the (potentially empty) set of atomic propositions common to all states from F,[3]

$$L'(s) = \begin{cases} L(s) & \text{if } s \in S \setminus Z \\ \bigcap\limits_{z \in Z} L(z) & \text{otherwise (i.e. if } s = \overline{z}) \end{cases}$$

Finally, for every reward structure defined over the Markov chain M, state $\overline{z}$ from the induced Markov chain is annotated with a reward

$$\overline{rwd} = \mathcal{R}^{\mathsf{rwd}}_{=?}[\mathrm{F}\ \mathsf{e}] \tag{6}$$

1. As shown later in this section, these new atomic propositions allow the computation of the reachability probabilities $\mathcal{P}_{=?}[\mathrm{F}\ \mathsf{z}]$ for the states $z \in Z_{\mathsf{out}}$ and of the reachability reward $\mathcal{R}^{\mathsf{rwd}}_{=?}[\mathrm{F}\ \mathsf{e}]$ for any reward structure rwd defined over F.

2. Note that $\sum_{s' \in S'} \mathbf{P}'(\overline{z}, s') = \sum_{s' \in S \setminus Z} \sum_{z \in Z_{\mathsf{out}}} prob_z \mathbf{P}(z, s') = \sum_{z \in Z_{\mathsf{out}}} prob_z \sum_{s' \in S \setminus Z} \mathbf{P}(z, s') = \sum_{z \in Z_{\mathsf{out}}} prob_z \cdot 1 = \mathcal{P}_{=?}[\mathrm{F}\ \bigvee_{z \in Z_{\mathsf{out}}} z] = 1$ as required.

3. Since the occurrence of $\overline{z}$ on a path of the abstract MC stands for *all paths through the fragment*, including any other atomic proposition *ap* would be equivalent to (invalidy) stating that *ap* is true in all states from F.

calculated over the parametric MC M_Z associated with F. Thus, $\overline{rwd}$ represents the cumulative reward to reach the end state of M_Z.

Example 3. Consider again the parametric Markov chain from our running example (Fig. 3a). The corresponding abstract MC induced by fragment F_1 from Example 1 is shown in Fig. 3c(i). This abstract MC is obtained by replacing all the states from F_1 with the single abstract state $\overline{z}_1$, and by using the rules from Definition 6 to find the outgoing transition probabilities and atomic propositions for $\overline{z}_1$. For example, the transition probability from $\overline{z}_1$ to s_4 is calculated as:

$$\begin{aligned}\mathbf{P}'(\overline{z}_1, s_4) &= \sum_{z\in\{s_2,s_3\}} prob_z \cdot \mathbf{P}(z, s_4)\\ &= prob_{s_2}\cdot \mathbf{P}(s_2,s_4) + prob_{s_3}\cdot\mathbf{P}(s_3,s_4)\\ &= prob_{s_2}\cdot x + prob_{s_3}\cdot 0\\ &= prob_{s_2}\cdot x,\end{aligned}$$

where $prob_{s_2}= \mathcal{P}_{=?}[\mathrm{F s_2}]$ and $prob_{s_3}= \mathcal{P}_{=?}[\mathrm{F s_3}] = 1-prob_{s_2}$ are reachability properties calculated over the parametric MC associated with fragment F_1 (cf. Fig. 3b). As the two output-state reachability probabilities (5) for fragment F_1 can be expressed in terms of a single probability, we use the notation $prob_1 = prob_{s_2}$ for this probability in Fig. 3c(i). The transition probabilities from $\overline{z}_1$ to s_9 and s_{13} are calculated similarly, and the transition probability from s_{12} to $\overline{z}_1$ is simply $\mathbf{P}'(s_{12},\overline{z}_1) = \mathbf{P}(s_{12}, s_0) = y$ (since the entry state of F_1 is s_0). All other transition probabilities from $\overline{z}_1$ to other states and from other states to $\overline{z}_1$ are zero. State $\overline{z}_1$ is labelled with the atomic proposition op_1, which is the only label common to all states from the fragment F_1. Finally, $\overline{z}_1$ is annotated with the rewards $time_1 = \mathcal{R}^{\mathsf{time}}_{=?}[\mathrm{F}\ s_2 \vee s_3]$ and $cost_1 = \mathcal{R}^{\mathsf{cost}}_{=?}[\mathrm{F}\ s_2 \vee s_3]$ computed over the parametric MC associated with F_1.

Fig. 3c(ii) shows the abstract Markov chain obtained after all three fragments F_1–F_3 from Example 1 were used to simplify the initial MC from Fig. 3a. Note how even for the small MC from our running example, the abstract MC from Fig. 3c(ii) is much simpler than the initial MC from Fig. 3a; the abstract MC has only 5 states and 10 transitions, compared to 15 states and 25 transitions for the initial MC.

The ePMC computation of unbounded until properties $\mathcal{P}_{=?}[\Phi_1\,\mathrm{U}\,\Phi_2]$ (and thus also of reachability properties $\mathcal{P}_{=?}[\mathrm{F}\ \Phi] = \mathcal{P}_{=?}[\mathit{true}\ \mathrm{U}\ \Phi]$) is based on the following result.[4]

Theorem 1. Let F be a fragment of a parametric Markov chain M, and Φ_1 and Φ_2 two PCTL state formulae over M. If every atomic proposition ap that appears in Φ_1 or Φ_2 either holds in every fragment state z (i.e., $ap \in L(z)$) or holds in no such state,[5] then the PMC of the until PCTL formula $\mathcal{P}_{=?}[\Phi_1\,\mathrm{U}\,\Phi_2]$ over M yields an expression equivalent to that produced by the PMC of the formula over the abstract Markov chain induced by F.

4. The computation of bounded until properties is not supported because path lengths are not preserved by the ePMC abstraction process.

5. While this requirement restricts the state formulae that the theorem applies to (or, alternatively, the fragments that can be usefully constructed), the evaluation presented later in the paper shows that the theorem supports the efficient analysis of multiple properties of practical relevance.

Proof. Let A, A' be the sets of paths that satisfy the PCTL formula $\Phi_1\mathrm{U}\,\Phi_2$ for the MC $M(S, s_0, \mathbf{P}, L)$ and for the MC $M'(S', s'_0, \mathbf{P}', L')$ induced by $F(Z, z_0, Z_{\mathsf{out}})$:

$$A = \{\pi \in \mathit{Paths}^M(s_0)\,|\,\exists i > 0.(\pi(i) \models \Phi_2 \wedge \forall j < i.\pi(j) \models \Phi_1)\}$$

$$A' = \{\pi' \in \mathit{Paths}^{M'}(s'_0)\,|\,\exists i > 0.(\pi'(i) \models \Phi_2 \wedge \forall j < i.\pi'(j) \models \Phi_1)\}$$

According to the semantics of PCTL (cf. Section 2.2), we need to show that $\mathrm{Pr}_{s_0}(A) = \mathrm{Pr}'_{s'_0}(A')$, where Pr_{s_0} and $\mathrm{Pr}'_{s'_0}$ are probability measures defined over $\mathit{Paths}^M(s_0)$ and $\mathit{Paths}^{M'}(s'_0)$, respectively, as explained in Section 2.1. A path $\pi \in A$ has the general form

$$\pi = \pi_0\omega_1\pi_1\omega_2\pi_2\ldots\omega_n\pi_n\ldots, \tag{7}$$

where $\pi_0, \pi_1, \ldots, \pi_n$ are subpaths comprising only states from $S \setminus Z$, $\omega_1, \omega_2, \ldots, \omega_n$ are subpaths comprising only states from Z, and the first state of π that satisfies Φ_2 is the last state from the path prefix $\pi_0\omega_1\pi_1\omega_2\pi_2\ldots\omega_n\pi_n$. Note that (7) subsumes the scenarios when the path prefix starts with a state from Z (subpath π_0 empty), ends with a state from Z (subpath π_n empty), contains only states from $S \setminus Z$ ($n = 0$), or contains only states from Z ($n = 1$, and subpaths π_0 and π_1 empty). We have three cases.

Case 1) If $n > 0$ and π_n is non-empty, the subpaths ω_1, ω_2, …, ω_n must all start with z_0 and end with a state $z \in Z_{\mathsf{out}}$ (as paths can only "enter" and "exit" MC fragments through their only entry state and one of their output states, respectively). Furthermore, z_0 and z must satisfy Φ_1 (since it belongs to the prefix of path π), so all states from Z also satisfy Φ_1 (as required by the theorem). This means that substituting any subset of ω_1, ω_2, …, ω_n from π with any combination of fragment subpaths starting at z_0 and ending with a state from Z_{out} changes π into a sequence of states that either belongs to A (if it is a path from $\mathit{Paths}^M(s_0)$) or has probability 0 of occurring in M (otherwise). Given the set of all paths $X(\pi_0, \pi_1, \ldots, \pi_n) \subseteq A$ that can be obtained through such substitutions, and using the notation π_i^{last} for the last state on subpath π_i, $1 \le i \le n$, we have:

$$\begin{aligned}&\mathrm{Pr}_{s_0}(X(\pi_0,\pi_1,\ldots,\pi_n)) =\\ &\left[\prod_{i=0}^{n-1}\mathbf{P}(\pi_i)\mathbf{P}(\pi_i^{\mathsf{last}}, z_0)\sum_{z\in Z_{\mathsf{out}}} prob_z\mathbf{P}(z,\pi_{i+1}(1))\right]\mathbf{P}(\pi_n) =\\ &\qquad\left[\prod_{i=0}^{n-1}\mathbf{P}'(\pi_i)\mathbf{P}'(\pi_i^{\mathsf{last}},\overline{z})\mathbf{P}'(\overline{z},\pi_{i+1}(1))\right]\mathbf{P}'(\pi_n) =\\ &\qquad\qquad\mathrm{Pr}'_{s'_0}(X'(\pi_0,\pi_1,\ldots,\pi_n))\end{aligned}$$

where $X'(\pi_0, \pi_1, \ldots, \pi_n) = \{\pi' \in \mathit{Paths}^{M'}(s'_0)\,|\,\pi' = \pi_0\overline{z}\pi_1\ \overline{z}\pi_2\ldots\overline{z}\pi_n\ldots\} \subseteq A'$ (since all states $\pi_0, \pi_1, \ldots, \pi_n$ and $\overline{z}$ satisfy Φ_1, and π_n^{last} satisfies Φ_2). We can similarly show that any subset $X'(\pi_0, \pi_1, \ldots, \pi_n)$ of A' with paths of the form $\pi_0\overline{z}\pi_1\ \overline{z}\pi_2\ldots\overline{z}\pi_n\ldots$ and on which Φ_2 first holds in state π_n^{last} corresponds to a subset $X(\pi_0, \pi_1, \ldots, \pi_n) \subseteq A$ such that $\mathrm{Pr}'_{s'_0}(X'(\pi_0, \pi_1, \ldots, \pi_n)) = \mathrm{Pr}_{s_0}(X(\pi_0, \pi_1, \ldots, \pi_n))$.

Case 2) If $n > 0$ and π_n is empty, then the last state of ω_n from (7) satisfies Φ_2, and (since one state from Z satisfies Φ_2 and the theorem requires that every atomic proposition from Φ_2 either is true for all states from Z or is false for all states from Z), all states from Z must satisfy Φ_2. This includes the states from ω_1 and its initial state, z_0, so the set of paths π form the set $X(\pi_0) = \{\pi \in Paths^M(s_0) \mid \pi = \pi_0 z_0 \ldots\} \subseteq A$, and we have:

$$\mathrm{Pr}_{s_0}(X(\pi_0)) = \mathbf{P}(\pi_0)\mathbf{P}(\pi_0^{\mathsf{last}}, z_0) = \\ \mathbf{P}'(\pi_0)\mathbf{P}'(\pi_0^{\mathsf{last}}, \overline{z}) = \mathrm{Pr}'_{s'_0}(X'(\pi_0)),$$

where $X'(\pi_0) = \{\pi' \in Paths^{M'}(s'_0) \mid \pi' = \pi_0\overline{z}\ldots\} \subseteq A'$. In a similar way, we can show that any subset $A'(\pi_0)$ of A' with paths of the form $\pi_o\overline{z}\ldots$ and on which Φ_2 first holds in state $\overline{z}$ corresponds to a subset $X(\pi_0)$ of A such that $\mathrm{Pr}'_{s'_0}(X'(\pi_0)) = \mathrm{Pr}_{s_0}(X(\pi_0))$.

Case 3) Finally, if $n = 0$, the path π from (7) becomes $\pi = \pi_0\ldots$, and equiprobable sets $X(\pi_0) \subseteq A$ and $X'(\pi_0) \subseteq A'$ are straightforward to identify.

We have shown that A and A' can be partitioned into pairs of corresponding subsets $X \subseteq A$ and $X' \subseteq A'$ that are equiprobable according to the probability metrics Pr_{s_0} and $\mathrm{Pr}'_{s'_0}$, which completes the proof. □

The repeated application of Theorem 1 reduces the computation of until properties $\mathcal{P}_{=?}[\Phi_1 \,\mathrm{U}\, \Phi_2]$ of a parametric MC with multiple fragments F_1, F_2, ... to computing:

1) the output-state reachability probabilities for the parametric MCs associated with F_1, F_2, ...;
2) $\mathcal{P}_{=?}[\Phi_1 \,\mathrm{U}\, \Phi_2]$ for the parametric MC induced by the fragments,

and combining the results from the two ePMC stages into a set of algebraic formulae over the parameters of the original MC. The parametric MCs from these stages are typically much simpler than the original, "monolithic" MC, and much faster to analyse. In addition, ePMC focuses on frequently used domain-specific fragments F_1, F_2, ..., and thus stage 1 only needs to be executed once for a domain. Note that a result similar to Theorem 1 is not available for bounded until properties $\mathcal{P}_{=?}[\Phi_1 \,\mathrm{U}^{\leq k}\, \Phi_2]$ because the abstract MC induced by a set of fragments $F_1, F_2, \ldots$ does not preserve the path lengths from the original MCs.

Example 4. We use the above two-stage method to compute properties P_1 and P_2 from our running example (cf. Table 1).

In stage 1 we compute the output-state reachability properties for the parametric MCs associated with fragments F_1–F_3 (cf. Fig. 3b):

- for the MC associated with F_1, $prob_1 = p_{11} + (1-p_{11})p_{12}$;
- for the MC associated with F_2, $prob_2 = \alpha_1 p_{21} + \alpha_2 p_{22}$;
- for the MC associated with F_3, $prob_3 = (p_{31} + (1-p_{31})\, p_{32}) / (1-(1-p_{31})(1-p_{32})r)$.

We computed these algebraic expressions manually, based on the MCs from Fig. 3b. However, they can also be obtained using one of the model checkers mentioned earlier (i.e., PARAM, PRISM and Storm), or can be taken directly from our ePMC repository of such expressions for the service-based systems domain (see Section 6 later in the paper).

In stage 2, we use a probabilistic model checker (we used Storm) to compute P_1 and P_2 over the induced parametric MC from Fig. 3c. The shaded formulae from Table 2 show the expressions obtained for P_1 and P_2, preceded by the results from the first ePMC stage.

TABLE 2
ePMC of the QoS properties from the running example

Output-state reachability formulae computed in stage 1 of ePMC	
$prob_1 = p_{11} + (1-p_{11})p_{12}$	$prob_2 = \alpha_1 p_{21} + \alpha_2 p_{22}$
$prob_3 = \frac{p_{31}+(1-p_{31})p_{32}}{1-(1-p_{31})(1-p_{32})r}$	
$time_1 = t_{11} + (1-p_{11})t_{12}$	$cost_1 = c_{11} + (1-p_{11})c_{12}$
$time_2 = \alpha_1 t_{21} + \alpha_2 t_{22}$	$cost_2 = \alpha_1 c_{21} + \alpha_2 c_{22}$
$time_3 = \frac{t_{31}+(1-p_{31})t_{32}}{1-(1-p_{31})(1-p_{32})r}$	$cost_3 = \frac{c_{31}+(1-p_{31})c_{32}}{1-(1-p_{31})(1-p_{32})r}$

Property-specific formulae computed in stage 2 of ePMC

Prop.	**PCTL formula**	**ePMC set of formulae**
P_1	$\mathcal{P}_{=?}[\mathsf{F\ succ}]$	$P_1 = \frac{prob_1(xprob_2+(1-x)(1-y)prob_3)}{1-(1-x)yprob_1prob_3}$
P_2	$\mathcal{P}_{=?}[\neg\mathsf{op}_3 \ \mathsf{U\ fail}]$	$P_2 = 1 - prob_1 + xprob_1(1-prob_2)$
T	$\mathcal{R}^{\mathsf{time}}_{=?}[\mathsf{F\ succ} \vee \mathsf{fail}]$	$T = \frac{time_1+prob_1(xtime_2+(1-x)time_3)}{1-(1-x)yprob_1prob_3}$
C	$\mathcal{R}^{\mathsf{cost}}_{=?}[\mathsf{F\ succ} \vee \mathsf{fail}]$	$C = \frac{cost_1+prob_1(xcost_2+(1-x)cost_3)}{1-(1-x)yprob_1prob_3}$

Expectedly, the set of formulae from Table 2 is much simpler than the "monolithic" P_1 and P_2 formulae from Table 1. As we will show in Section 8, this difference is even more significant for larger models, making the computation and evaluation of "monolithic" formulae challenging for existing PMC techniques.

The final result from this section allows the efficient parametric model checking of reachability reward properties.

Theorem 2. Let F be a fragment of a parametric Markov chain M, and T a set of states from M. If T includes no state from F, then the PMC of the reachability reward PCTL formula $\mathcal{R}_{=?}[\mathrm{F}\, T]$ over M yields an expression equivalent to that produced by the PMC of the formula over the abstract MC M' induced by F.

Proof. We adopt one of the alternative definitions for the reachability reward $\mathcal{R}_{=?}[\mathrm{F}\, T]$ from [3, §10.5.1]. Given the set A of all finite paths $\pi = s_0 s_1 \ldots s_m$ from M such that $s_m \in T$ and $s_0, s_1, \ldots, s_{m-1} \notin T$, we have:

$$\mathcal{R}_{=?}[\mathrm{F}\, T] = \begin{cases} 0, & \text{if } \mathcal{P}_{<1}[\mathrm{F}\, T] \\ \sum_{\pi \in A} \mathbf{P}(\pi)\rho(\pi), & \text{otherwise} \end{cases} \tag{8}$$

where $\rho(\pi) = \sum_{i=0}^{m-1} \rho(s_i)$.

According to Theorem 1, if $\mathcal{P}_{<1}[\mathrm{F}\, T]$ over M then $\mathcal{P}_{<1}[\mathrm{F}\, T]$ over M' too, so $\mathcal{R}_{=?}[\mathrm{F}\, T] = 0$ over both M and M' and the theorem holds. The theorem also holds trivially when A contains no paths with states from Z: in this case, it is straightforward to show that A is also the set of all finite paths $s_0 s_1 \ldots s_m$ from M' such that $s_m \in T$ and $s_0, s_1, \ldots, s_{m-1} \notin T$. As such, the rest of the proof considers the case when $\mathcal{P}_{\geq 1}[\mathrm{F}\, T]$ and A contains paths with states from Z. The first state from Z on such a path will be z_0 (the only entry state of the fragment F), immediately followed by other states from Z until a state from Z_{out} is reached, and then followed by at least one state from $S \setminus Z$ (since

$T \cap Z = \emptyset$ and Z_{out} states are followed by a state from outside Z). Thus, the generic form of such a path is

$$\pi = \underbrace{\overbrace{s_0 \ldots s_i}^{\in S\setminus Z} \overbrace{s_{i+1}}^{=z_0} \overbrace{s_{i+2} \ldots s_{j-1}}^{\in Z} \overbrace{s_j}^{\in Z_{\text{out}}} \overbrace{s_{j+1}}^{\in S\setminus Z} \overbrace{s_{j+2} \ldots s_{m-1}}^{\in S}}_{\notin T} \underbrace{s_m}_{\in T} \tag{9}$$

We consider the subset of all paths $A_1 \subseteq A$ that start with the prefix $s_0 \ldots s_i z_0$ and, using the notation $\pi_{x,y} = s_x s_{x+1} \ldots\ s_y$, we calculate their contribution to the sum from the second row of (8):

$$\begin{aligned}&C(s_0, s_1, \ldots, s_i, z_0) = \textstyle\sum_{\pi \in A_1} \mathbf{P}(\pi)\rho(\pi) = \\ &\textstyle\sum_{\pi \in A_1} \mathbf{P}(\pi_{0,i+1})\mathbf{P}(\pi_{i+1,m})(\rho(\pi_{0,i+1}) + \rho(\pi_{i+1,m})) = \\ &\mathbf{P}(\pi_{0,i+1}) \left[\left(\textstyle\sum_{\pi \in A_1} \mathbf{P}(\pi_{i+1,m})\right) \rho(\pi_{0,i+1}) + \right. \\ &\qquad\qquad \left.\textstyle\sum_{\pi \in A_1} \mathbf{P}(\pi_{i+1,m})\rho(\pi_{i+1,m})\right] = \\ &\mathbf{P}(\pi_{0,i+1}) \left[1 \cdot \rho(\pi_{0,i+1}) + \textstyle\sum_{\pi \in A_1} \mathbf{P}(\pi_{i+1,m})\rho(\pi_{i+1,m})\right],\end{aligned}$$

since $\mathcal{P}_{\geq 1}[\mathrm{F}\, T]$ requires that $\sum_{\pi \in A_1} \mathbf{P}(\pi_{i+1,m}) = 1$. Additionally, using the shorthand notation $\pi_{z_0 \to z}$ and $\pi_{j+1 \to T}$ for the set of all sub-paths $s_{i+1}s_{i+2} \ldots s_j$ associated with a fixed $s_j = z \in Z_{\text{out}}$ and for the set of all sub-paths $s_{j+1}s_{j+2} \ldots s_m$ from (9), respectively, we have:

$$\begin{aligned}&\textstyle\sum_{\pi \in A_1} \mathbf{P}(\pi_{i+1,m})\rho(\pi_{i+1,m}) = \\ &\textstyle\sum_{z \in Z_{\text{out}}} \sum_{\pi_1 \in \pi_{z_0 \to z}} \sum_{\pi_2 \in \pi_{j+1 \to T}} \mathbf{P}(\pi_1)\mathbf{P}(z, s_{j+1})\mathbf{P}(\pi_2) \\ &\qquad\qquad (\rho(\pi_1) + \rho(z) + \rho(\pi_2)) = \\ &\textstyle\sum_{z \in Z_{\text{out}}} \sum_{\pi_1 \in \pi_{z_0 \to z}} \left[\mathbf{P}(\pi_1)(\rho(\pi_1) + \rho(z)) \cdot \right. \\ &\qquad\qquad \left.\textstyle\sum_{\pi_2 \in \pi_{j+1 \to T}} \mathbf{P}(z, s_{j+1})\mathbf{P}(\pi_2)\right] + \\ &\textstyle\sum_{z \in Z_{\text{out}}} \left[\left(\sum_{\pi_1 \in \pi_{z_0 \to z}} \mathbf{P}(\pi_1)\right) \cdot \right. \\ &\qquad\qquad \left.\textstyle\sum_{\pi_2 \in \pi_{j+1 \to T}} \mathbf{P}(z, s_{j+1})\mathbf{P}(\pi_2)\rho(\pi_2))\right] = \\ &\textstyle\sum_{z \in Z_{\text{out}}} \sum_{\pi_1 \in \pi_{z_0 \to z}} \left[\mathbf{P}(\pi_1)(\rho(\pi_1) + \rho(z)) \cdot 1\right] + \\ &\textstyle\sum_{z \in Z_{\text{out}}} \left[\mathit{prob}_z \cdot \sum_{\pi_2 \in \pi_{j+1 \to T}} \mathbf{P}(z, s_{j+1})\mathbf{P}(\pi_2)\rho(\pi_2)\right] = \\ &\overline{\mathit{rwd}} + \textstyle\sum_{\pi_2 \in \pi_{j+1 \to T}} \left(\sum_{z \in Z_{\text{out}}} \mathit{prob}_z \mathbf{P}(z, s_{j+1})\right) \mathbf{P}(\pi_2)\rho(\pi_2) = \\ &\overline{\mathit{rwd}} + \textstyle\sum_{\pi_2 \in \pi_{j+1 \to T}} \mathbf{P}'(\overline{z}, s_{j+1})\mathbf{P}(\pi_2)\rho(\pi_2),\end{aligned}$$

where $\overline{\mathit{rwd}}$, prob_z and $\mathbf{P}'(\overline{z}, s_{j+1})$ are those from Definition 6, and where we used the fact that $\mathcal{P}_{\geq 1}[\mathrm{F}\, T]$ to infer that $\sum_{\pi_2 \in \pi_{j+1 \to T}} \mathbf{P}(z, s_{j+1})\mathbf{P}(\pi_2) = 1$ and to include all non-zero-probability sub-paths at every step of the calculation. Combining the results so far, we obtain

$$\begin{aligned}&C(s_0, s_1, \ldots, s_i, z_0) = \mathbf{P}(\pi_{0,i+1}) \left(\rho(\pi_{0,i+1}) + \overline{\mathit{rwd}} + \right. \\ &\qquad\qquad \left.\textstyle\sum_{\pi_2 \in \pi_{j+1 \to T}} \mathbf{P}'(\overline{z}, s_{j+1})\mathbf{P}(\pi_2)\rho(\pi_2)\right).\end{aligned}$$

This reward "contribution" is equivalent to that obtained by replacing all sub-paths from F appearing in paths from A_1 immediately after the prefix $s_0 s_1 \ldots s_i$ with state $\overline{z}$ comprising a reward value $\overline{\mathit{rwd}}$ and transition probabilities $\mathbf{P}'(\overline{z}, s_{j+1})$ to states $s_{j+1} \in S \setminus Z$. By repeating this process to replace all occurrences of sub-paths from F appearing in A, we obtain an expression equivalent to $\sum_{\pi \in A} \mathbf{P}(\pi)\rho(\pi)$, but corresponding to the parametric model checking of $\mathcal{R}_{=?}[\mathrm{F}\, T]$ over the abstract MC M' induced by F, which completes the proof. □

The previous theorem reduces the computation of reachability reward properties $\mathcal{R}_{=?}[\mathrm{F}\, T]$ of a parametric MC with fragments F_1, F_2, ... to computing:

1) the per-fragment cumulative reachability reward properties for the MCs associated with F_1, F_2, ...;
2) $\mathcal{R}_{=?}[\mathrm{F}\, T]$ for the MC induced by these fragments,

and combining the results from the two stages into a set of algebraic formulae over the parameters of the original MC. Note that results similar to Theorem 2 are not available for instantaneous, cumulative and steady-state rewards formulae because the abstract MC induced by F_1, F_2, ... does not preserve the path lengths and the rewards structures of the original MC.

Example 5. We use ePMC to calculate properties *time* and *cost* from our running example (cf. Table 1), starting with the cumulative reachability reward properties for fragments F_1–F_3, i.e., $\overline{t}_i$ and $\overline{c}_i$ for $i = 1, 2, 3$. The resulting formulae, which we obtained manually (but which can also be obtained using a PMC tool) are shown in the top half of Table 2. For the second ePMC stage, we used the model checker Storm to obtain the algebraic expressions for *time* and *cost* from the lower half of Table 2.

As in Example 4, ePMC produced a set of formulae that is far simpler than the "monolithic" *time* and *cost* from Table 1. Note that we do not compare the analysis time of our ePMC method with that of existing PMC here or in Example 4 because for the simple system from our running example the two analysis times are similar. However, we do provide an extensive comparison of these analysis times for larger systems in our evaluation of ePMC from Section 8.

5 Implementation

We developed a *pattern-aware parametric model checker* that implements the theoretical results from the previous section. This tool automates the second stage of ePMC. As shown in Fig. 1, the ePMC tool uses a domain-specific repository of QoS-property expressions to analyse PCTL-specified QoS properties of a parametric Markov chain annotated with pattern instances.

The domain-specific repository comprises entries with the general format:

```
pattern_name(parameter_list):
  property_name=expr, ..., property_name=expr;
```

Each such entry defines algebraic expressions for the reachability properties and the reachability reward properties of a parametric MC fragment commonly used within the domain of interest, i.e., a *modelling pattern*.

Example 6. Table 3 shows a part of the ePMC repository for the service-based systems domain. This part includes the three patterns used by the operations from our running example (SEQ for op$_1$, PROB for op$_2$, and SEQ_R for op$_3$), such that the formulae from the top half of Table 2 can be obtained (without any calculations) by instantiating the relevant patterns:

$$\begin{aligned}\mathit{prob}_1 &= \mathsf{SEQ}(p_{11}, c_{11}, t_{11}, p_{12}, c_{12}, t_{12}).\mathsf{prob} \\ \mathit{prob}_2 &= \mathsf{PROB}(\alpha_1, p_{21}, c_{21}, t_{21}, \alpha_2, p_{22}, c_{22}, t_{22}).\mathsf{prob} \\ \mathit{prob}_3 &= \mathsf{SEQ\_R}(p_{31}, c_{31}, t_{31}, p_{32}, c_{32}, t_{32}, r).\mathsf{prob} \\ \mathit{time}_1 &= \mathsf{SEQ}(p_{11}, c_{11}, t_{11}, p_{12}, c_{12}, t_{12}).\mathsf{time} \\ \mathit{cost}_1 &= \mathsf{SEQ}(p_{11}, c_{11}, t_{11}, p_{12}, c_{12}, t_{12}).\mathsf{cost} \\ \mathit{time}_2 &= \mathsf{PROB}(\alpha_1, p_{21}, c_{21}, t_{21}, \alpha_2, p_{22}, c_{22}, t_{22}).\mathsf{time} \\ \mathit{cost}_2 &= \mathsf{PROB}(\alpha_1, p_{21}, c_{21}, t_{21}, \alpha_2, p_{22}, c_{22}, t_{22}).\mathsf{cost} \\ \mathit{time}_3 &= \mathsf{SEQ\_R}(p_{31}, c_{31}, t_{31}, p_{32}, c_{32}, t_{32}, r).\mathsf{time} \\ \mathit{cost}_3 &= \mathsf{SEQ\_R}(p_{31}, c_{31}, t_{31}, p_{32}, c_{32}, t_{32}, r).\mathsf{cost}\end{aligned} \tag{10}$$

TABLE 3
Fragment of the ePMC repository of QoS-property expressions for the service-based systems domain, comprising expressions for the probability of pattern invocation success `prob`, expected `cost`, and expected execution `time`

```
SEQ(p1,c1,t1,p2,c2,t2):
 prob=p1+(1−p1)*p2, cost=c1+c2*(1−p1),
 time=t1+(1−p1)*t2;
 ...
PROB(x1,p1,c1,t1,x2,p2,c2,t2):
 prob=x1*p1+x2*p2, cost=x1*c1+x2*c2,
 time=x1*t1+x2*t2;
 ...
SEQ_R(p1,c1,t1,p2,c2,t2,r):
 prob=(p1+(1−p1)*p2)/(1−(1−p1)*(1−p2)*r),
 cost=(c1+(1−p1)*c2)/(1−(1−p1)*(1−p2)*r),
 time=(t1+(1−p1)*t2)/(1−(1−p1)*(1−p2)*r);
 ...
```

The ePMC model checker supports the analysis of pattern-annotated parametric Markov chains specified in the PRISM high-level modelling language. This language models a system as the parallel composition of a set of *modules*. The state of a *module* is encoded by a set of finite-range local variables, and its state transitions are defined by probabilistic guarded commands that change these variables, and have the general form:

$$[action]\ guard \rightarrow e_1{:}\ update_1 + \ldots + e_N{:}\ update_N;$$

In this command, *guard* is a boolean expression over all model variables. If *guard* evaluates to *true*, the arithmetic expression e_i, $1 \leq i \leq N$, gives the probability with which the $update_i$ change of the module variables occurs. When the label *action* is present, all modules comprising commands with this *action* have to synchronise (i.e., can only carry out one of these commands simultaneously).

Example 7. Fig. 4 shows how the parametric MC from Fig. 3c(ii) and its reward structures and labels are specified in this high-level modelling language. The values z = 1 to z = 5 of the local variable z from the Workflow module correspond to states $\overline{z}_1$, $\overline{z}_2$, $\overline{z}_3$, s_{13} and s_{14} from Fig. 3c(ii), respectively. The model parameters prob1 to prob3, cost1 to cost3 and time1 to time3 are associated with the pattern annotations from the lines starting with a triple forward slash '///'. These annotations tell the ePMC model checker that the model has parameters associated with QoS properties of modelling patterns from the repository in Table 3. For example, the shaded pattern annotation at the top of Fig. 4 specifies that the QoS properties prob, cost and time of the SEQ modelling pattern appear as parameters named prob1, cost1 and time1 in the model, where the id '1' is provided before the pattern name. The occurrences of these parameters are also shaded in Fig. 4.

The general format of an ePMC pattern annotation is:

```
/// id: pattern_name(actual_parameter_list)
```

This annotation indicates that some or all QoS properties of the pattern appear as parameters in the parametric MC, with the name of each such parameter obtained by appending the id from the annotation to the name of the QoS property.

```
/// 1: SEQ(p11,c11,t11,p12,c12,t12)
/// 2: PROB(alpha1,p21,c22,t21,alpha2,p22,c22,t22)
/// 3: SEQ_R(p31,c31,t31,p32,c32,t32,r)

const double x;
const double y;
const double prob1;
const double cost1;
const double time1;
...

module Workflow
  z : [1..5] init 1;
  [ ] z=1 → x*prob1:(z'=2) + (1-x)*prob1:(z'=3) + (1-prob1):(z'=4);
  [ ] z=2 → prob2:(z'=5) + (1-prob2):(z'=4);
  [ ] z=3 → y*prob3:(z'=1) + (1-y)*prob3:(z'=5) + (1-prob3):(z'=4);
  [ ] z=4 → true;
  [ ] z=5 → true;
endmodule

rewards "cost"
  z=1: cost1;
  z=2: cost2;
  z=3: cost3;
endrewards

rewards "time"
  z=1: time1;
  z=2: time2;
  z=3: time3;
endrewards

label "success" = (z=5);
label "fail" = (z=4);
label "op3" = (z=3);
```

Fig. 4. Pattern-annotated parametric Markov chain for the service-based system from the running example

Given a domain-specific repository of QoS-property expressions, a pattern-annotated parametric MC and a set of PCTL-encoded QoS properties, the ePMC model checker uses the theoretical results from Section 4 to compute a set of algebraic formulae comprising:

1. Formulae for every MC parameter associated with a modelling pattern listed in the model annotations. These formulae are obtained by instantiating the expressions from the repository, as shown in (10). The top half of Table 2 shows these formulae for the MC in Fig. 4.
2. A formula for each analysed QoS property, obtained by applying standard parametric model checking, i.e., by ignoring the pattern annotations of the parametric MC. The bottom half of Table 2 shows these formulae for the four QoS properties from our running example.

The ePMC tool can be configured to use PRISM or Storm for the computation of the latter formulae, and outputs the combined set of algebraic formulae as a MATLAB file, ready for evaluation or further analysis with MATLAB.

6 ePMC of Service-Based Systems

Service-based systems (SBSs) enable the effective development of new applications through the integration of third-party and in-house components implemented as services. SBSs are widely used, including in business-critical applications from e-commerce, online banking and e-government,

TABLE 4
Modelling patterns for the implementation of SBS operations using n functionally-equivalent services

Pattern	Description
$\mathsf{SEQ}(p_1, c_1, t_1, \ldots, p_n, c_n, t_n)$	The n services are invoked in order, stopping after the first successful invocation or after the last service.
$\mathsf{PAR}(p_1, c_1, t_1, \ldots, p_n, c_n, t_n)^\dagger$	The n services are all invoked at the same time, and the operation uses the first result returned by a successful invocation (if any).
$\mathsf{PROB}(x_1, p_1, c_1, t_1, \ldots, x_n, p_n, c_n, t_n)$	A single service is invoked; x_i gives the probability that this is service i, where $\sum_{i=1}^n x_i = 1$.
$\mathsf{SEQ\_R}(p_1, c_1, t_1, \ldots, p_n, c_n, t_n, r)$	The n services are invoked in order as for the SEQ pattern; if all n invocations fail, the execution of the operation is retried (from service 1) with probability r or the operation fails with probability $1 - r$.
$\mathsf{SEQ\_R1}(p_1, c_1, t_1, r_1, \ldots, p_n, c_n, t_n, r_n)$	The services are invoked in order. If service i fails, it is reinvoked with probability r_i; with probability $1 - r_i$, the operation is attempted using service $i + 1$ (if $i < n$) or fails (if $i = n$).
$\mathsf{PAR\_R}(p_1, c_1, t_1, \ldots, p_n, c_n, t_n, r)^\dagger$	All n services are invoked as for the PAR pattern; if all n invocations fail, the execution of the operation is retried with probability r or the operation fails with probability $1 - r$.
$\mathsf{PROB\_R}(x_1, p_1, c_1, t_1, \ldots, x_n, p_n, c_n, t_n, r)$	Like for PROB, a single service i is invoked; if the invocation fails, the PROB pattern is retried with probability r or the operation fails with probability $1 - r$.
$\mathsf{PROB\_R1}(x_1, p_1, c_1, t_1, r_1, \ldots, x_n, p_n, c_n, t_n, r_n)$	Like for PROB, a single service i is invoked; however, its invocation is retried after failure(s) with probability r_i or the operation fails with probability $1 - r_i$.

†Pattern unsuitable for non-idempotent operations (e.g. credit card payment in an e-commerce SBS)

and evolve frequently as a result of maintenance or self-adaptation. This evolution often requires the QoS analysis of alternative SBS implementations which deliver the same functionality, to select an implementation that meets the QoS requirements of the system. The alternative SBS implementations differ in the way in which they use the multiple functionally-equivalent services that are available for each of their operations. Given $n \geq 1$ services that can perform the same SBS operation with probabilities of success p_1, p_2, $\ldots$, p_n, costs $c_1, c_2, \ldots, c_n$, and execution times $t_1, t_2, \ldots, t_n$, the operation can be implemented using one of the patterns from the (potentially non-exhaustive) pattern set described in Table 4. As we discuss further in Section 9, variants of the first three patterns have been widely used in related research (e.g. in [6], [14], [42]), while—to the best of our knowledge—the remaining patterns from Table 4 have not been considered before.

As indicated earlier in the paper, our ePMC method is well suited for the SBS domain, as the operation implementation patterns from Table 4 correspond to component modelling patterns whose reliability, cost and execution time can be obtained in the first stage of the method. Table 5 and the following theorem provide the repository of (manually derived) closed-form expressions for all patterns from Table 4 and three key QoS properties of SBS operations.

Theorem 3. The closed-form expressions from Table 5 specify the success probability, the expected cost, and the expected execution time for each of the SBS-operation implementation patterns from Table 4.

Proof. We prove the SEQ results by induction. For the base case, we have $n = 1$, corresponding to an SBS operation carried out by a single service with success probability p_1, cost c_1 and execution time t_1. As required, $p_{\mathsf{SEQ}} = p_1 = 1 - (1 - p_1)$, $c_{\mathsf{SEQ}} = c_1$ and $t_{\mathsf{SEQ}} = t_1$. Assume now that the SEQ expressions from Table 5 are correct for n services, and consider an SEQ pattern comprising $n+1$ services. There are two ways in which the $n+1$ services can complete the operation successfully: (i) either the first n services complete the operation successfully (with probability $1 - \prod_{i=1}^n (1 - p_i)$), or (ii) each of the first n services fails, and the invocation of the $(n + 1)$-th service is successful. Accordingly, the success probability for the $(n + 1)$-service SEQ pattern is:

$$(1-\textstyle\prod_{i=1}^n (1-p_i)) + ((\prod_{i=1}^n (1-p_i))\, p_{n+1}) = 1 - \prod_{i=1}^{n+1} (1-p_i).$$

To calculate the expected cost and execution time for the $(n+1)$-service SEQ pattern, recall that the $(n+1)$-th service is invoked iff the invocations of all previous n services failed, i.e. with probability $\prod_{i=1}^n (1 - p_i)$. Accordingly, using the $(n + 1)$-th service adds a supplementary expected cost of $(\prod_{i=1}^n (1 - p_i))\, c_{n+1}$ and a supplementary expected execution time of $(\prod_{i=1}^n (1 - p_i))\, t_{n+1}$ to the expected cost and execution time of an n-service SEQ pattern, respectively. Thus, the expected cost for the $(n + 1)$-service SEQ pattern is given by

$$\left(c_1 + \textstyle\sum_{i=2}^n \left(\prod_{j=1}^{i-1} (1-p_j)\right) c_i\right) + (\prod_{i=1}^n (1-p_i))\, c_{n+1} = \\ = c_1 + \textstyle\sum_{i=2}^{n+1} \left(\prod_{j=1}^{i-1} (1-p_j)\right) c_i$$

and the expected execution time can be calculated similarly, which completes the induction step.

For the PAR pattern, the probability that the parallel invocations of the n (independent) services will all fail is $\prod_{i=1}^n (1 - p_i)$, so the probability that the operation will be completed successfully is $1 - \prod_{i=1}^n (1 - p_i)$ as required. Also, since all n services are always invoked, the cost for the pattern is given by the sum of the n service costs. Finally, to calculate the expected execution time for the PAR pattern, assume (as stated in Table 5 and without loss of generality) that the n services are ordered such that $t_1 \leq t_2 \leq \cdots \leq t_n$. Under this assumption, service i will be the first service that completes execution *successfully* (in time t_i) iff: (i) the invocations of the faster services $1, 2, \ldots, i - 1$ have all failed (which happens with probability $\prod_{j=1}^{i-1} (1 - p_j)$); and (ii) the invocation of service i is successful (which happens with probability p_i). Thus, the execution time for the PAR patterns follows a discrete distribution with probability $\left(\prod_{j=1}^{i-1} (1 - p_j)\right) p_i$ of successful completion in time t_i, and

TABLE 5
Complete ePMC repository of QoS-property expressions for the SBS domain

Pattern	**Success probability**	**Expected cost**	**Expected execution time**
SEQ	$p_{\mathsf{SEQ}} = 1 - \prod_{i=1}^{n}(1-p_i)$	$c_{\mathsf{SEQ}} = c_1 + \sum_{i=2}^{n}\left(\prod_{j=1}^{i-1}(1-p_j)\right)c_i$	$t_{\mathsf{SEQ}} = t_1 + \sum_{i=2}^{n}\left(\prod_{j=1}^{i-1}(1-p_j)\right)t_i$
PAR[†]	$p_{\mathsf{PAR}} = p_{\mathsf{SEQ}}$	$c_{\mathsf{PAR}} = \sum_{i=1}^{n} c_i$	$t_{\mathsf{PAR}} = \widetilde{p}_1 t_1 + \sum_{i=2}^{n}\left(\prod_{j=1}^{i-1}(1-p_j)\right)\widetilde{p}_i t_i$
PROB	$p_{\mathsf{PROB}} = \sum_{i=1}^{n} x_i p_i$	$c_{\mathsf{PROB}} = \sum_{i=1}^{n} x_i c_i$	$t_{\mathsf{PROB}} = \sum_{i=1}^{n} x_i t_i$
SEQ_R	$p_{\mathsf{SEQ\_R}} = \frac{p_{\mathsf{SEQ}}}{1-(1-p_{\mathsf{SEQ}})r}$	$c_{\mathsf{SEQ\_R}} = \frac{c_{\mathsf{SEQ}}}{1-(1-p_{\mathsf{SEQ}})r}$	$t_{\mathsf{SEQ\_R}} = \frac{t_{\mathsf{SEQ}}}{1-(1-p_{\mathsf{SEQ}})r}$
SEQ_R1[‡]	$p_{\mathsf{SEQ\_R1}} = 1 - \prod_{i=1}^{n}(1-p'_i)$	$c_{\mathsf{SEQ\_R1}} = c'_1 + \sum_{i=2}^{n}\left(\prod_{j=1}^{i-1}(1-p'_j)\right)c'_i$	$t_{\mathsf{SEQ\_R1}} = t'_1 + \sum_{i=2}^{n}\left(\prod_{j=1}^{i-1}(1-p'_j)\right)t'_i$
PAR_R	$p_{\mathsf{PAR\_R}} = p_{\mathsf{SEQ\_R}}$	$c_{\mathsf{PAR\_R}} = \frac{c_{\mathsf{PAR}}}{1-(1-p_{\mathsf{PAR}})r}$	$t_{\mathsf{PAR\_R}} = \frac{t_{\mathsf{PAR}}}{1-(1-p_{\mathsf{PAR}})r}$
PROB_R	$p_{\mathsf{PROB\_R}} = \frac{p_{\mathsf{PROB}}}{1-(1-p_{\mathsf{PROB}})r}$	$c_{\mathsf{PROB\_R}} = \frac{c_{\mathsf{PROB}}}{1-(1-p_{\mathsf{PROB}})r}$	$t_{\mathsf{PROB\_R}} = \frac{t_{\mathsf{PROB}}}{1-(1-p_{\mathsf{PROB}})r}$
PROB_R1[‡]	$p_{\mathsf{PROB\_R1}} = \sum_{i=1}^{n} x_i p'_i$	$c_{\mathsf{PROB\_R1}} = \sum_{i=1}^{n} x_i c'_i$	$t_{\mathsf{PROB\_R1}} = \sum_{i=1}^{n} x_i t'_i$

[†]assuming that the n services are ordered such that $t_1 \leq t_2 \leq \cdots \leq t_n$, with $\widetilde{p}_i = p_i$ for $i < n$ and $\widetilde{p}_n = 1$
[‡] $p'_i = \frac{p_i}{1-(1-p_i)r_i}$, $c'_i = \frac{c_i}{1-(1-p_i)r_i}$ and $t'_i = \frac{t_i}{1-(1-p_i)r_i}$ for all $i = 1, 2, \ldots, n$

probability $\prod_{j=1}^{n}(1-p_j)$ of unsuccessful completion in time t_n. As a result, the expected execution time for the pattern is given by:

$$\begin{aligned}
&\textstyle\sum_{i=1}^{n}\left(\prod_{j=1}^{i-1}(1-p_j)\right)p_i t_i + \left(\prod_{j=1}^{n}(1-p_j)\right)t_n = p_1 t_1 + \\
&+\textstyle\sum_{i=2}^{n-1}\left(\prod_{j=1}^{i-1}(1-p_j)\right)p_i t_i + \left[\left(\prod_{j=1}^{n-1}(1-p_j)\right)p_n t_n + \right.\\
&\left.+\left(\prod_{j=1}^{n-1}(1-p_j)\right)(1-p_n)t_n\right] = \\
&= p_1 t_1 + \textstyle\sum_{i=2}^{n-1}\left(\prod_{j=1}^{i-1}(1-p_j)\right)p_i t_i + \left(\prod_{j=1}^{n-1}(1-p_j)\right)t_n,
\end{aligned}$$

which can be easily rearranged in the format from Table 5 by introducing the notation $\widetilde{p}_i = p_i$ for $i < n$ and $\widetilde{p}_n = 1$.

For the PROB pattern, the results from Table 5 follow immediately from the fact that the success probability, cost and execution time of the operation have a discrete distribution with probabilities $x_1, x_2, \ldots, x_n$ of taking the values $p_1, p_2, \ldots, p_n$ (for the success probability), $c_1, c_2, \ldots, c_n$ (for the cost), and $t_!, t_2, \ldots, t_n$ (for the execution time).

For the SEQ_R1 pattern, we first focus on a single service i with success probability p_i, cost c_i and execution time t_i. If unsuccessful invocations of the service (which happen with probability $(1-p_i)$) are followed by its re-invocation with probability r_i, then the overall probability of successfully invoking the service is given by:

$$\begin{aligned}
p'_i &= p_i + (1-p_i)r_i\Big(p_i + (1-p_i)r_i(p_i + \ldots)\Big) = \\
&= p_i + [(1-p_i)r_i]p_i + [(1-p_i)r_i]^2 p_i + \ldots = \\
&= \lim_{m\to\infty}\textstyle\sum_{j=0}^{m}[(1-p_i)r_i]^j p_i = \\
&= \lim_{m\to\infty} p_i \frac{1-[(1-p_i)r_i]^{m+1}}{1-[(1-p_i)r_i]} = \frac{p_i}{1-(1-p_i)r_i}.
\end{aligned} \tag{11}$$

A similar reasoning can be used to show that the expected cost and execution time of the service with re-invocations are

$$c'_i = \frac{c_i}{1-(1-p_i)r_i} \quad \text{and} \quad t'_i = \frac{t_i}{1-(1-p_i)r_i}, \tag{12}$$

respectively. Thus, the n-service SEQ_R1 pattern from Table 5 is equivalent to an n-service SEQ pattern whose services have success probabilities p'_1, p'_2, …, p'_n, costs c'_1, c'_2, …, c'_n, and execution times t'_1, t'_2, …, t'_n. As a result, the expressions for the success probability, expected cost and expected execution time of the SEQ_R1 pattern can be obtained by using these parameters in the analogous expressions of the SEQ pattern, as shown in Table 5.

The SEQ_R pattern is equivalent to having a single service with success probability p_{SEQ}, cost c_{SEQ} and execution time t_{SEQ}, and re-invoking this service with probability r after unsuccessful invocations. As such, the success probability, expected cost and expected execution time for the SEQ_R pattern are obtained by applying the formulae from (11) and (12) to this equivalent service, which yields the expressions from Table 5.

Using the same reasoning as for the SEQ_R pattern, it is straightforward to show that Table 5 provides the correct expressions for the PAR_R and PROB_R patterns.

Finally, the n-service PROB_R1 pattern is equivalent to an n-service PROB pattern whose i-th service has success probability p'_i given by (11), and cost c'_i and execution time t'_i given by (12). Using these three formulae as parameters in the expressions giving the success probability, expected cost and expected execution time of the PROB pattern yields the results for the PROB_R1 pattern. □

As we show experimentally in Section 8, ePMC can use the repository of closed-form expressions from Table 5 to efficiently compute reliability, cost and response-time QoS properties of realistic SBS designs that leading model checkers take a very long time to verify, or cannot handle at all due to out-of-memory or timeout errors. Furthermore, as also shown in Section 8, our method yields closed-form expressions that are more compact and take far less time to evaluate than the expressions produced by traditional PMC.

7 ePMC of Multi-Tier Architectures

As a second application domain for ePMC, we consider the deployment of software systems with a multi-tier architecture on a set of servers. For improved reliability and throughput, these systems often use *horizonal distribution* within some or all tiers, i.e. they have instances of these tiers running on multiple servers. In this section, we devise a repository of *server modelling patterns* for analysing reliability properties of such systems. To this end, we consider an m-tier software system comprising $n_1, n_2, \ldots, n_m \geq 1$ instances of tiers 1 through m, and we assume that these tier instances are deployed across multiple servers of different types.

Our (non-exhaustive) set of server modelling patterns is presented in Table 6. The BASIC pattern from this table corresponds to a server whose failure leads to the immediate loss of all tier instances running on the server, while the VIRTUALIZED and VIRTUALIZED-M(onitored) patterns correspond to servers where each instance of a tier is running within a separate virtual machine (VM). The difference between the two types of virtualized server is that the second type has a *monitor* component capable of detecting imminent server failures early enough to allow the migration of the VMs to other servers.

We assume that the engineers responsible for deploying a multi-tier software system on a combination of such servers need to assess the following reliability properties of alternative deployment options:

1) The probability P_{FAIL} of system failure due to all instances of a tier failing within a time period of interest;
2) The probability P_{SPF} of a single point of failure (i.e. a tier with a single operational instance) occurring within the analysed time period.

ePMC can support the analysis of these properties by using a repository of QoS-property expressions comprising entries for each probability $p_{b_1,b_2,\ldots,b_m}$ that $b_i \in \{0,1,2+\}$ instances of tier i, $i \in \{1,2,\ldots,m\}$, remain operational on the types of server from Table 6 at the end of the analyzed time period. For example, $p_{0,2+}$ represents the probability that a server running instances of two tiers at the beginning of the analysed period is left with no instance of the first tier and with two or more instances of the second tier at the end of the period. Although 3^m expressions need to be computed for these probabilities and each type of server, this is feasible because m is a small number (e.g., $m \leq 3$ for a three-tier architecture).

Table 7 and the following theorem provide the repository of (manually derived) closed-form expressions for all server modelling patterns from Table 6.

Theorem 4. The $p_{b_1,b_2,\ldots,b_m}$ expressions from Table 7 specify the probabilities that $b_1, b_2, \ldots, b_m$ instances of tiers $1, 2, \ldots, m$ remain operational on a BASIC, VIRTUALIZED and VIRTUALIZED-M server, respectively.

Proof. For the BASIC pattern, either the server remains operational and all $n_1, n_2, \ldots, n_m$ tier instances are still running at the end of the analysed time period, or the server fails and no instance is left running. The former scenario occurs with probability p, so $p_{b_1,b_2,\ldots,b_m} = p$ iff $b_i = 1$ for all tiers i for which $n_i = 1$ and $b_i = 2+$ for all tiers i for which $n_i > 1$; and the latter scenario occurs with probability $1-p$, so $p_{0,0,\ldots,0} = 1-p$. Otherwise, $p_{b_1,b_2,\ldots,b_m} = 0$ since there is no scenario in which only some of the tier instances are left running and others are lost.

For the VIRTUALIZED pattern, we first consider the scenario where at least one of $b_1, b_2, \ldots, b_m$ is non-zero. This requires that $m+1$ independent events occur: server stays up; and the appropriate number of VMs running instances of tier $i \in \{1,2,\ldots,m\}$ (i.e. zero, one, or greater than one) remain operational. The probability of the first event is p, and the probability of each of the other events is given by the probability that the value of a random variable with binomial distribution $\mathcal{B}(n_i, p_{\mathsf{VM}})$ is: zero (i.e. $(1-p_{\mathsf{VM}})^{n_i} = f(0,n_i)$); one (i.e. $n_i p_{\mathsf{VM}}(1-p_{\mathsf{VM}})^{n_i-1} = f(1,n_i)$); or greater than one (i.e. $1 - f(0,n_i) - f(1,n_i) = f(2+,n_i)$). The first part of the result from Table 7 is obtained by multiplying these $m+1$ probabilities. Finally, the scenario $b_1 = b_2 = \cdots = b_m = 0$ occurs in two circumstances: when the server fails—which happens with probability $(1-p)$, and when the server stays up but all $n_1 + n_2 + \cdots n_m$ VMs running tier instances fail—which happens with probability $p(1-p_{\mathsf{VM}})^{\sum_{i=1}^m n_i}$, giving the last part of the result for the VIRTUALIZED pattern.

Finally, for the VIRTUALIZED-M pattern, the scenario where at least one of b_1, b_2, …, b_m is non-zero can occur in two cases. In the first case, the server stays up and the appropriate number of VMs from each tier remains operational, which has probability $p\prod_{i=1}^m f(b_i,n_i)$ (as shown above for the VIRTUALIZED pattern); this corresponds to the first term from the definition of $p_{b_1,b_2,\ldots,b_m}$ from Table 7. In the second case, the server fails but the failure is detected (which happens with probability $(1-p)p_{\mathsf{detect}}$), and b_i VMs running tier i are successfully migrated to other servers and remain operational for $i = 1,2,\ldots,m$. To prove that the second term from the definition of $p_{b_1,b_2,\ldots,b_m}$ is correct, we will show that the probability of this last event is $g(b_i,n_i)$ from Table 7 for all three values of b_i. We start by noting that a given VM from tier i is successfully migrated with probability $\frac{p_{\mathsf{migrate}}}{1-(1-p_{\mathsf{migrate}}r)}$, a result that can be obtained as in (11); so the VM will be migrated *and* remain operational with probability $\frac{p_{\mathsf{migrate}}p_{\mathsf{VM}}}{1-(1-p_{\mathsf{migrate}}r)}$. Accordingly, $g(0,n_i)$, $g(1,n_i)$ and $g(2+,n_i)$ represent the probabilities that a random variable with binomial distribution $\mathcal{B}\Big(n_i, \frac{p_{\mathsf{migrate}}p_{\mathsf{VM}}}{1-(1-p_{\mathsf{migrate}}r)}\Big)$ takes values 0, 1, or greater than or equal to 2, respectively, which corresponds to the definition of $g(b_i,n_i)$ from Table 7. We complete the proof by noting that the scenario where $b_1 = b_2 = \cdots = b_m = 0$ occurs in the same two cases, as well as when the server fails and its failure is not detected (which happens with probability $(1-p)(1-p_{\mathsf{detect}})$, and corresponds to the last term from the definition of $p_{b_1,b_2,\ldots,b_m}$ for this scenario). ☐

Example 8. We consider a three-tier system adapted from [13], [35], and comprising client, business and data management tiers. We assume that there are two instances of each tier, and that these instances are deployed on four servers as shown in Fig. 5. To compute systems of closed-

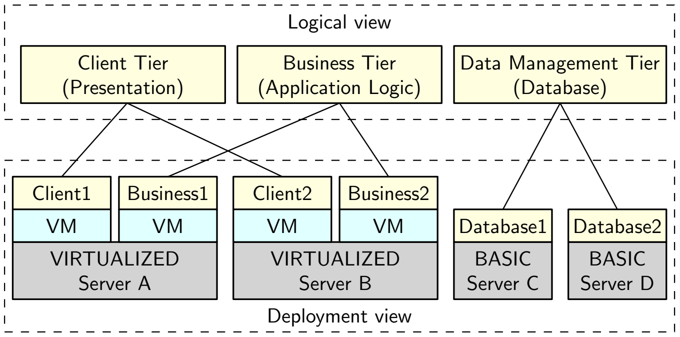


Fig. 5. Three-tier system deployed across four servers

TABLE 6
Server modelling patterns for the reliability analysis of multi-tier architecture deployments

Pattern	**Description**
BASIC$(n_1, n_2, \ldots, n_m, p)$	The $n_1, n_2, \ldots, n_m$ tier instances are deployed on a server whose probability of remaining operational throughout a time period of interest (e.g. a month) is p; if the server fails, all tier instances are lost.
VIRTUALIZED$(n_1, n_2, \ldots, n_m, p, p_{\mathsf{VM}})$	Each of the $n_1, n_2, \ldots, n_m$ tier instances runs within its own virtual machine (VM) on a server whose probability of remaining operational during the time period of interest is p; additionally, each VM has a probability p_{VM} of remaining operational during the same time period, independently of the other VMs.
VIRTUALIZED-M$(n_1, n_2, \ldots, n_m, p, p_{\mathsf{detect}}, p_{\mathsf{migrate}}, r, p_{\mathsf{VM}})$	The $n_1, n_2, \ldots, n_m$ tier instances are deployed within different VMs (each with probability p_{VM} of not failing), on a server whose probability of remaining operational is p. If the server does fail, there is a probability p_{detect} that a software monitor will detect the approaching failure before it happens, allowing the VMs to be migrated to other servers. The migration of each VM succeeds with probability p_{migrate}, and is retried with probability r in case of failure.

TABLE 7
ePMC repository of QoS-property expressions for the multi-tier architecture domain

Pattern	**Probability that $b_i \in \{0, 1, 2+\}$ tier-i instances, $i \in \{1, 2, \ldots, m\}$, remain operational**
BASIC	$p_{b_1,b_2,\ldots,b_m} = \begin{cases} p, & \text{if } \forall i \in \{1,2,\ldots,m\}.(n_i > 1 \wedge b_i = 2+) \vee (n_i = 1 \wedge b_i = 1) \\ 1-p, & \text{if } b_1 = b_2 = \ldots = b_m = 0 \\ 0, & \text{otherwise} \end{cases}$
VIRTUALIZED	$p_{b_1,b_2,\ldots,b_m} = \begin{cases} p\prod_{i=1}^m f(b_i, n_i), & \text{if } \exists i \in \{1,2,\ldots,m\}.b_i \neq 0 \\ p\prod_{i=1}^m f(b_i, n_i) + (1-p), & \text{if } b_1 = b_2 = \ldots = b_m = 0 \end{cases}$
VIRTUALIZED-M	$p_{b_1,b_2,\ldots,b_m} = \begin{cases} p\prod_{i=1}^m f(b_i, n_i) + (1-p)p_{\mathsf{detect}}\prod_{i=1}^m g(b_i, n_i), & \text{if } \exists i \in \{1,2,\ldots,m\}.b_i \neq 0 \\ p\prod_{i=1}^m f(b_i, n_i) + (1-p)p_{\mathsf{detect}}\prod_{i=1}^m g(b_i, n_i) + (1-p)(1-p_{\mathsf{detect}}), & \text{if } b_1 = b_2 = \ldots = b_m = 0 \end{cases}$
with $f(b_i,n_i)=\begin{cases} (1-p_{\mathsf{VM}})^{n_i}, & \text{if } b_i = 0 \\ n_i p_{\mathsf{VM}}(1-p_{\mathsf{VM}})^{n_i-1}, & \text{if } b_i = 1 \\ 1 - f(0,n_i) - f(1,n_i), & \text{if } b_i = 2+ \end{cases}$	; $g(b_i,n_i)=\begin{cases} \left(\frac{(1-p_{\mathsf{migrate}})(1-r)+p_{\mathsf{migrate}}(1-p_{\mathsf{VM}})}{1-(1-p_{\mathsf{migrate}})r}\right)^{n_i}, & \text{if } b_i = 0 \\ n_i \frac{p_{\mathsf{migrate}}p_{\mathsf{VM}}}{1-(1-p_{\mathsf{migrate}})r}\left(\frac{(1-p_{\mathsf{migrate}})(1-r)+p_{\mathsf{migrate}}(1-p_{\mathsf{VM}})}{1-(1-p_{\mathsf{migrate}})r}\right)^{n_i-1}, & \text{if } b_i = 1 \\ 1 - g(0,n_i) - g(1,n_i), & \text{if } b_i = 2+ \end{cases}$

form expressions for the reliability properties P_{FAIL} and P_{SPF} introduced earlier in this section, we built the pattern-annotated Markov chain from Fig. 6.

The transient states of this MC are organised into four stages (separated by dashed vertical lines in the diagram), where the states from each stage and their outgoing transitions model the effect of possible failures associated with one of the servers from Fig. 5. For example, stage 1 corresponds to server A. This stage has a single state, which is labelled '2, 2, 2' to indicate that, before considering failures on any of the servers, the system has precisely two active instances within each tier. The four outgoing transitions of this state correspond to the possible outcomes for the two tier instances on server A: neither instance fails; only the business-tier instance Business1 fails; only the client-tier instance Client1 fails; or both tier instances fail. Stage 2 corresponds to server B, and its four initial states (which reflect the four possible outcomes of stage 1) have outgoing transitions that model the failures which may occur on server B. The outcomes due to possible failures on servers C and D are modelled in a similar way by the other two stages of the MC. The MC states within the four stages are annotated with the relevant server modelling patterns, i.e., VIRTUALIZED$(1, 1, p^A, p^A_{\mathsf{VM}})$ for server A, VIRTUALIZED$(1, 1, p^B, p^B_{\mathsf{VM}})$ for server B, BASIC$(1, p^C)$ for server C, and BASIC$(1, p^D)$ for server D.

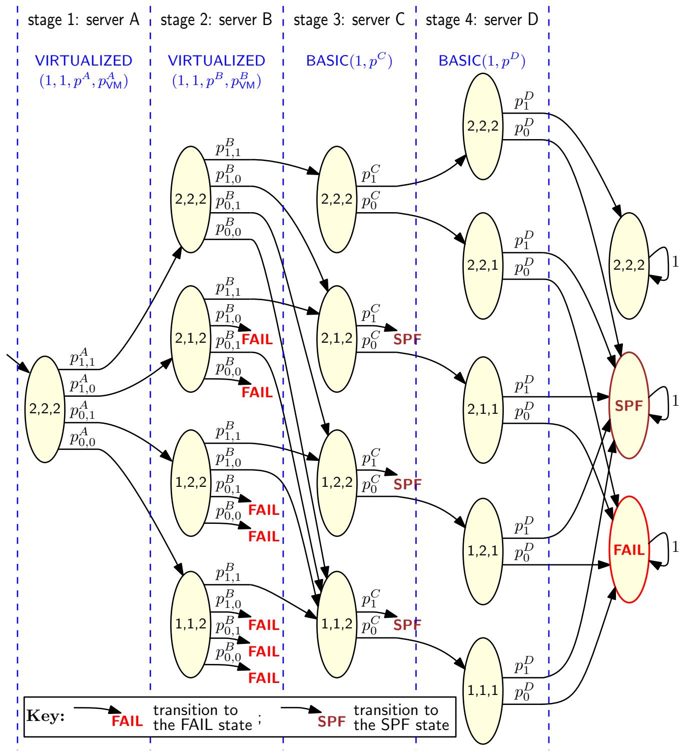

stage 1: server A
stage 2: server B
stage 3: server C
stage 4: server D
VIRTUALIZED $(1,1,p^A,p^A_{\mathsf{VM}})$
VIRTUALIZED $(1,1,p^B,p^B_{\mathsf{VM}})$
BASIC$(1,p^C)$
BASIC$(1,p^D)$
2,2,2
2,1,2
1,2,2
1,1,2
2,2,1
2,1,1
1,2,1
1,1,1
SPF
FAIL
Key: FAIL transition to the FAIL state ; SPF transition to the SPF state
Fig. 6. Pattern-annotated Markov chain for the three-tiered system deployment from Fig. 5

TABLE 8
Probability of failure (P_{FAIL}) and probability of single point of failure (P_{SPF}) for the three-tier system from Fig. 5

$$P_{\mathsf{FAIL}} = p^A_{0,0}p^B_{1,1}p^C_0 p^D_0 + p^A_{0,0} + p^A_{0,1}p^B_{1,0}p^C_0 p^D_0 + p^A_{1,1}p^B_{0,0}p^C_0 p^D_0 + p^A_{0,1}p^B_{1,1}p^C_0 p^D_0 + p^A_{1,1}p^B_{1,1}p^C_0 p^D_0 + p^A_{1,1}p^B_{1,0}p^C_0 p^D_0 + p^A_{0,1}p^B_{0,1} + p^A_{1,0}p^B_{1,1}p^C_0 p^D_0 + p^A_{1,0}p^B_{1,0} + p^A_{1,0}p^B_{0,0} + p^A_{0,1}p_{0,0}B + p^A_{1,1}p^B_{0,1}p^C_0 p^D_0 + p^A_{1,0}p^B_{0,1}p^C_0 p^D_0 - p^A_{0,0}p^B_{1,1}$$

$$P_{\mathsf{SPF}} = p^A_{0,0}p^B_{1,1}p^C_0 p^D_1 + p^A_{0,1}p^B_{1,0}p^C_0 p^D_1 + p^A_{1,0}p^B_{0,1}p^C_1 + p^A_{1,0}p^B_{0,1}p^C_0 p^D_1 + p^A_{0,1}p^B_{1,0}p1C + p^A_{1,1}p^B_{0,0}p1C + p^A_{0,1}p^B_{1,1}p^C_0 p^D_1 + p^A_{1,1}p^B_{0,1}p^C_1 + p^A_{1,1}p^B_{1,1}p^C_1 p^D_0 + p^A_{1,1}p^B_{1,1}p^C_0 p^D_1 + p^A_{1,1}p^B_{1,0}p1C + p^A_{1,1}p^B_{1,0}p^C_0 p^D_1 + p^A_{1,0}p^B_{1,1}p^C_0 p^D_1 + p^A_{0,1}p^B_{1,1}p^C_1 + p^A_{1,0}p^B_{1,1}p1C + p^A_{1,1}p^B_{0,1}p^C_0 p^D_1 + p^A_{1,1}p^B_{0,0}p^C_0 p^D_1 + p^A_{0,0}p^B_{1,1}p^C_1$$

$p^A_{1,1} = p^A(p^A_{\mathsf{VM}})^2$	$p^A_{1,0} = p^A p^A_{\mathsf{VM}}(1-p^A_{\mathsf{VM}})$
$p^A_{0,1} = p^A p^A_{\mathsf{VM}}(1-p^A_{\mathsf{VM}})$	$p^A_{0,0} = (1-p^A) + p^A(1-p^A_{\mathsf{VM}})^2$
$p^B_{1,1} = p^B(p^B_{\mathsf{VM}})^2$	$p^B_{1,0} = p^B p^B_{\mathsf{VM}}(1-p^B_{\mathsf{VM}})$
$p^B_{0,1} = p^B p^B_{\mathsf{VM}}(1-p^B_{\mathsf{VM}})$	$p^B_{0,0} = (1-p^B) + p^B(1-p^B_{\mathsf{VM}})^2$
$p^C_1 = p^C \quad p^C_0 = 1-p^C$	$p^D_1 = p^D \quad p^D_0 = 1-p^D$

To keep the model simple, all states with zero instances within at least one tier are joined together into a single state (labelled 'FAIL'); and all non-FAIL states reached after modelling all four servers and comprising a single instance within at least one tier are combined into a "single point of failure" ('SPF') state. The other MC states are labelled 'x, y, z', to denote the presence of x client-tier instances, y business-tier instances and z database-tier instances. FAIL and SPF are absorbing states, as is a state labelled '$2, 2, 2$' and corresponding to no failures occurring in the system.

Given this pattern-annotated parametric MC and the repository from Table 7, ePMC computes the set of formulae from Table 8 for the properties $P_{\mathsf{FAIL}} = \mathcal{P}_{=?}[\mathsf{F\ FAIL}]$ and $P_{\mathsf{SPF}} = \mathcal{P}_{=?}[\mathsf{F\ SPF}]$. The first two formulae from this table were obtained using Storm [21] to verify the parametric MC from Fig. 6, and the other formulae were obtained from the repository in Table 7.

8 Evaluation

We carried out extensive experiments to compare the feasibility, scalability and efficiency of ePMC to those of the model checkers PRISM, Storm and PARAM. All experiments were performed on a Ubuntu-16 server with i7-4770@3.40GHz × 8 processors and 16GB of memory, on which we installed the latest versions of the three model checkers downloaded from their websites and our ePMC pattern-aware parametric model checker from Section 5. To ensure the reproducibility of our results, we made the models and verified properties from our experiments available on the ePMC website.

To also assess the generality of our method, we evaluated it for both the service-based systems domain introduced in Section 6 and the multi-tier architectures domain presented in Section 7. The experimental results for these two domains are reported in the next two sections, followed by a discussion of the threats to the validity of our results in Section 8.3.

8.1 Service-based systems domain

We evaluated ePMC by using it to analyse a six-component service-based system initially introduced in [27] and also used in [8], [26]. This system implements a workflow used to carry out foreign exchange (FX) trading transactions as illustrated by the UML activity diagram in Fig. 7. Traders can use the FX system in "normal" or "expert" operation modes. In its normal mode, the system uses a Fundamental Analysis component to decide whether a transaction should be performed, or the fundamental analysis should be retried, or the normal-mode session should be ended. Performing a transaction involves using an Order component to carry out the operation, and is followed by the invocation of a Notification component to inform the trader about the outcome of the transaction. In its expert mode, the system uses a Market Watch component to obtain exchange market data that is then processed by a Technical Analysis component. The result of this analysis may satisfy a set of trader-specified objectives (in which case a transaction is performed), may not meet these objectives (so the Market Watch is reinvoked for an update) or may be erroneous (in which case an Alarm component is used to warn the trader). In our experiments, we assumed that the probabilities that annotate the decision points from the diagram in Fig. 7 (i.e., the operational profile of the FX system) were unknown parameters x, y_1, y_2, z_1 and z_2.

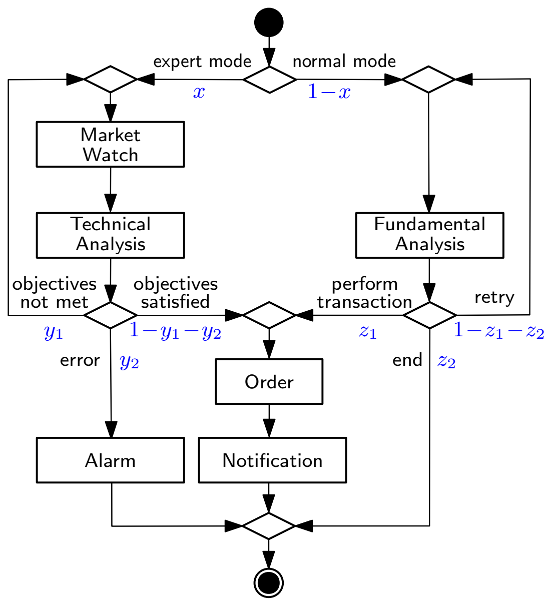


Fig. 7. Foreign exchange system from the SBS domain

To evaluate ePMC, we considered multiple ways in which the six FX components could be implemented using the SBS modelling patterns from Table 4 with between one and five functionally-equivalent services per component. To analyse this large set of alternative designs using ePMC, we developed a pattern-annotated parametric MC similar to the MC from Fig. 3c(ii) but modelling the FX system. To analyse the same designs with the model checkers PRISM, Storm and PARAM, we obtained an individual "monolithic" model for each design by using a dedicated parametric MC generator that we implemented for this purpose.

Three QoS properties of the system were analyzed: (P1) the probability of successful completion; (P2) the expected execution time; and (P3) the expected cost. Table 9 compares the PMC time required to produce the sets of

TABLE 9
Parametric model checking time (seconds) for the FX service-based system

		ePMC			PRISM			Storm			PARAM		
Pattern	**#services**	P1	P2	P3	P1	P2	P3	P1	P2	P3	P1	P2	P3
SEQ	1	0.33	0.34	0.33	0.32	8.10	8.51	0.06	0.077	0.075	0.012	0.862	0.883
SEQ	2	0.35	0.35	0.38	32.87	M	M	3.61	7.61	7.59	5.24	T	T
SEQ	3	0.34	0.34	0.34	M	-	-	T	T	T	T	-	-
SEQ	4	0.36	0.35	0.36	-	-	-	-	-	-	-	-	-
SEQ	5	0.34	0.34	0.34	-	-	-	-	-	-	-	-	-
PAR	2	0.33	0.34	0.34	8.75	M	M	1.69	8.54	3.92	5.14	T	T
PAR	3	0.33	0.36	0.34	M	-	-	T	T	T	T	-	-
PAR	4	0.48	0.35	0.35	-	-	-	-	-	-	-	-	-
PAR	5	0.34	0.35	0.35	-	-	-	-	-	-	-	-	-
PROB	2	0.34	0.34	0.35	M	M	M	0.46	0.89	0.87	T	T	T
PROB	3	0.35	0.35	0.34	-	-	-	2.81	4.34	4.30	-	-	-
PROB	4	0.36	0.35	0.35	-	-	-	48.23	50.08	50.33	-	-	-
PROB	5	0.35	0.37	0.35	-	-	-	545.33	395.12	387.27	-	-	-
SEQ_R	2	0.35	0.36	0.35	M	M	M	830.24	T	T	T	T	T
SEQ_R	3	0.37	0.36	0.36	-	-	-	-	-	-	-	-	-
SEQ_R	4	0.36	0.36	0.37	-	-	-	-	-	-	-	-	-
SEQ_R	5	0.36	0.37	0.35	-	-	-	-	-	-	-	-	-
SEQ_R1	2	0.36	0.35	0.37	M	M	M	T	T	T	T	T	T
SEQ_R1	3	0.35	0.35	0.37	-	-	-	-	-	-	-	-	-
SEQ_R1	4	0.35	0.36	0.37	-	-	-	-	-	-	-	-	-
SEQ_R1	5	0.36	0.39	0.36	-	-	-	-	-	-	-	-	-
PAR_R	2	0.35	0.36	0.36	M	M	M	566.18	T	T	T	T	T
PAR_R	3	0.36	0.36	0.36	-	-	-	-	-	-	-	-	-
PAR_R	4	0.35	0.37	0.38	-	-	-	-	-	-	-	-	-
PAR_R	5	0.34	0.37	0.35	-	-	-	-	-	-	-	-	-
PROB_R	2	0.37	0.35	0.38	M	M	M	70.27	68.90	69.42	T	T	T
PROB_R	3	0.37	0.36	0.35	-	-	-	T	T	T	-	-	-
PROB_R	4	0.35	0.36	0.37	-	-	-	-	-	-	-	-	-
PROB_R	5	0.37	0.37	0.36	-	-	-	-	-	-	-	-	-
PROB_R1	2	0.36	0.37	0.36	M	M	M	T	T	T	T	T	T
PROB_R1	3	0.35	0.36	0.37	-	-	-	-	-	-	-	-	-
PROB_R1	4	0.35	0.36	0.35	-	-	-	-	-	-	-	-	-
PROB_R1	5	0.36	0.36	0.36	-	-	-	-	-	-	-	-	-
20 random	min	0.31	0.31	0.31	T	T	T	2.25	2.79	1.12	T	T	T
combinations	max	0.54	0.34	0.37				520.45	817.58	661.80			
of 2/3-service	mean	0.33	0.32	0.32				52.12	88.41	67.18			
SEQ/PAR/PROB	stdev	0.06	0.01	0.01				112.93	178.01	148.62			

M=out of memory, T=timeout (no result returned within 15 minutes), −=experiment skipped as PMC of smaller model failed

formulae for these three properties using ePMC to the time required to produce a single formula for each property using PRISM, Storm and PARAM. With the exception of the last row, the results correspond to experiments in which every FX component used the same pattern (SEQ, PAR, etc.) and the same number of services.

Table 9 shows that the PMC time required to analyse the three properties using ePMC is always better, and typically orders of magnitude smaller, than the PMC times of PRISM, Storm and PARAM (except for the trivial case when a single service is used for each SBS component, cf. row 1). Moreover, the three tools ran out of memory or timed out when components used four (and sometimes even two or three) services, with the exception of the Storm analyses of the PROB pattern, which were all completed. The reason for this is that PROB is by far the SBS modelling pattern with the simplest QoS-property expressions, i.e., just linear combinations of the service parameters, as shown in Table 5. Note also that ePMC analysis times are almost identical irrespective of the property analysed and of the pattern and number of services used. This is because the time required to run our ePMC tool is dominated by the time used to read the files containing the ePMC repository of QoS-property expressions and the annotated MC, to start Storm and to parse the MC, all of which do not depend on the analysed property or the patterns from the model annotations.

The last row from Table 9 reports the minimum, maximum and mean PMC time and the standard deviation over 20 experiments in which the pattern and number of services (two or three) used for each component were chosen randomly and independently of those of the other components. We only used the patterns SEQ, PAR and PROB in these experiments so that at least Storm could complete the analysis, albeit with PMC times much longer than ePMC; PRISM and PARAM timed out in all 20 experiments.

To assess the efficiency of evaluating ePMC-generated expressions, we plotted graphs of the three QoS properties of the FX system using both the sets of formulae generated by our ePMC tool and the "monolithic" formulae generated by Storm (the best performing of the current model checkers in our experiments). Fig. 8 shows three such graphs, generated with Matlab. These graphs correspond to the following fixed values for the FX operational profile parameters (which can be obtained in practice from system logs): $x = 0.66$, $y_1 = 0.61$, $y_2 = 0.11$, $z_1 = 0.27$ and $z_2 = 0.53$. Table 10 shows that ePMC yields far smaller and more efficient to evaluate formulae than traditional PMC. The only system instance for which the Storm formula size and graph generation time are comparable to (but still larger than) those of ePMC corresponds to PROB, i.e., the simplest

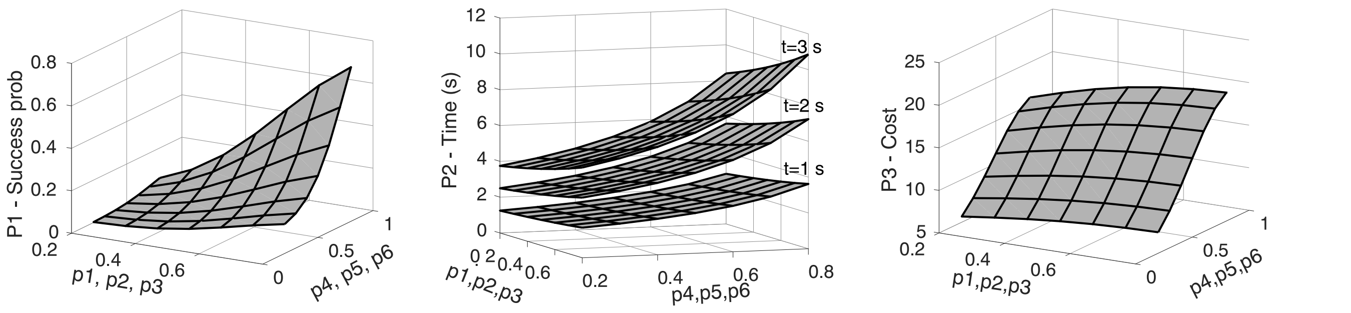


Fig. 8. QoS analysis showing increase in FX success probability and (as fewer FX sessions fail halfway) in expected execution time and cost when the reliability p_i of (all) services used for the i-th FX component, $1 \leq i \leq 6$, increases; for simplicity, all services are assumed to have the same mean execution time (1s, 2s or 3s) and the same cost of 1

TABLE 10
Comparison of ePMC and Storm formulae sizes and graph generation times for the graphs from Fig. 8 and system instances using the same SBS modelling pattern (with three services) for each FX component

Graph	SBS modelling pattern	ePMC formulae #operations	ePMC formulae time	Storm formula #operations	Storm formula time
P1	PROB_R	287	2.6s	285425	8.2 hours
P2	PROB	174	2.9s	9686	16.9s
P3	PAR	198	1.3s	171166	2.5 hours

SBS modelling pattern as discussed earlier in this section. For the other patterns from Table 10, the ePMC sets of formulae are several orders of magnitude smaller and faster to evaluate than the Storm formulae.

8.2 Multi-tier software architectures domain

We evaluated ePMC within this domain by using it to analyse the properties P_{FAIL} and P_{SPF} from Section 7 for eight four-server deployments of a three-tier system. The characteristics of these deployments and the time taken by the parametric model checking of the two properties are shown in Table 11; the timing information was collected through logging the output of the model checkers.

As for the SBS domain, the ePMC model checking time is largely unaffected by the system size, remaining under 1s when the total number of tier instances increases from six instances for deployments D1 and D2 to 40 instances for deployments D7 and D8. In contrast, the model checking time for PRISM, Storm and PARAM increases rapidly with the system size. As D1, D3, D5 and D7 use only the simpler, loop-free deployment patterns BASIC and VIRTUALIZED, the three model checkers can successfully analyse deployments D1, D3 and D5. However, the analysis times of these tools are already orders of magnitude larger than those of ePMC for the larger deployment D5 (and their analyses of deployment D7 time out). The better efficiency of ePMC is even clearer for deployments D2, D4, D6 and D8, which use the more complex deployment pattern VIRTUALIZED-M—out of these deployments, only D2 can be successfully analysed by PRISM, Storm and PARAM.

Finally, Table 12 shows the combined sizes of the P_{FAIL} and P_{SPF} formulae generated by ePMC and by the current model checkers for the deployments from Table 11. As for the SBS domain, the ePMC formulae are always smaller than those produced by the current model checkers. Moreover, they are over two orders of magnitude smaller for deployment D2, which is the only deployment that uses the more complex modelling pattern VIRTUALIZED-M and that PRISM, Storm and PARAM can analyse.

8.3 Threats to validity

External validity threats may arise if ePMC modelling patterns do not occur for other types of systems than those considered in our paper. To mitigate this threat, we evaluated ePMC for systems from two significantly different domains—service-based systems, and multi-tier software architectures. Furthermore, probabilistic model checking is increasingly used to analyse Markov chains comprising interchangeable modules within the important and broad domain of software product lines (e.g., [16], [29], [41]). These interchangeable modules represent ideal ePMC modelling pattern candidates, although further research is needed to confirm this hypothesis.

Construct validity threats may be due to the assumptions made when choosing and modelling the SBS and multi-tier software architecture systems from our evaluation. To mitigate these threats, we focused on systems, models and QoS properties adapted from previous case studies from the software engineering literature (e.g., [8], [13], [26], [27], [35]).

Internal validity threats can originate from how the experiments used to evaluate ePMC were performed, and from bias in the interpretation of the results. To address these threats, we carried out all the experiments on the same server (whose specification is provided at the beginning of Section 8); we used the latest probabilistic model checker versions available when we conducted the evaluation; and we made all models and experimental results publicly available on our project website in order to enable other researchers to replicate and verify our results.

9 Related work

Since its introduction in Daws' seminal work [19] in 2004, parametric model checking has underpinned the development of a vast array of methods for the modelling and analysis of software and other computer-based systems. These include methods for comparing alternative system designs

TABLE 11
Parametric model checking time (seconds or T=15-minute timeout) for eight deployments of a three-tier system

	Tier instances[†]				**Server type**[‡] | **Instances of tiers 1, 2, 3**				**ePMC**		**PRISM**		**Storm**		**PARAM**	
ID	**T1**	**T2**	**T3**	**Total**	**Server A**	**Server B**	**Server C**	**Server D**	P_{FAIL}	P_{SPF}	P_{FAIL}	P_{SPF}	P_{FAIL}	P_{SPF}	P_{FAIL}	P_{SPF}
D1[#]	2	2	2	6	V | 1,1,0	V | 1,1,0	B | 0,0,1	B | 0,0,1	0.26	0.26	0.39	0.15	0.55	0.10	0.06	0.03
D2	2	2	2	6	V-M | 1,1,0	V-M | 1,1,0	B | 0,0,1	B | 0,0,1	0.25	0.26	3.14	7.33	5.08	4.92	12.88	33.06
D3	4	4	2	10	V | 2,1,0	V | 2,1,0	V | 0,1,1	V | 0,1,1	0.26	0.26	2.50	1.86	1.00	1.01	0.53	0.53
D4	4	4	2	10	V-M | 2,1,0	V-M | 2,1,0	V-M | 0,1,1	V-M | 0,1,1	0.27	0.27	T	T	T	T	T	T
D5	8	8	4	20	V | 4,2,0	V | 4,2,0	V | 0,2,2	V | 0,2,2	0.29	0.31	742.85	1074.59	25.19	26.92	12.78	14.11
D6	8	8	4	20	V-M | 4,2,0	V-M | 4,2,0	V-M | 0,2,2	V-M | 0,2,2	0.29	0.30	T	T	T	T	T	T
D7	16	16	8	40	V | 8,4,0	V | 8,4,0	V | 0,4,4	V | 0,4,4	0.29	0.29	T	T	T	T	T	T
D8	16	16	8	40	V-M | 8,4,0	V-M | 8,4,0	V-M | 0,4,4	V-M | 0,4,4	0.30	0.30	T	T	T	T	T	T

[†] T1=Tier 1 instances; T2=Tier 2 instances; T3=Tier 3 instances
[‡] B=BASIC; V=VIRTUALIZED; V-M=VIRTUALIZED-M
[#] Deployment used in Example 8

TABLE 12
Combined size of P_{FAIL} and P_{SPF} formulae (#operations or T=timeout) for the parametric model checking experiments from Table 11

ID	**ePMC**	**PRISM**	**Storm**	**PARAM**
D1	143	204	258	240
D2	189	30584	34719	33667
D3	1688	1892	2234	2124
D4	1868	T	T	T
D5	9082	21952	25394	24248
D6	9404	T	T	T
D7	9086	T	T	T
D8	9412	T	T	T

[28], [29], sensitivity analysis [24], parameter synthesis [9], [20], [30], probabilistic model repair [4], [15], dynamic reconfiguration of self-adaptive systems [22], [23], and synthesis of confidence intervals for the QoS properties of software systems [10], [11]. These methods address very different problems, and yet most researchers who developed them mention the same limitation of parametric model checking: its computationally intensive nature. Addressing this one limitation can greatly improve the scalability and applicability of all the methods that use parametric model checking. Despite this significant incentive, research to improve PMC efficiency has been very limited so far. To the best of our knowledge, this research includes only the results from [32], [34]. As we explain in the rest of this section, these results represent significant advances, but are both complementary to our ePMC work.

The PMC technique presented in [32] provides major performance improvements over the initial PMC approach from [19]. While the language-theoretic PMC approach from [19] uses a regular expression to encode the probability that a PCTL path formula is satisfied, [32] computes a rational expression for the probability of reaching a set of parametric MC states, and mitigates the explosion in expression size relative to the number of MC states by exploiting algebraic symmetry and cancelation properties of rational functions. A further improvement introduced in [32] is the application of arithmetic operations during the state elimination stage of the PMC algorithm, to simplify the rational expression as it is calculated. The technique is implemented by the parametric model checkers PARAM [31] and PRISM [38], and shown to considerably reduce the complexity of PMC in [32]. Our work builds on the PMC technique from [32] (when using the parametric model checking functionality of PRISM in the second ePMC stage, cf. Section 5). Furthermore, ePMC complements the results from [32] by further speeding up parametric model checking through the pre-computation of PMC expressions for domain-specific modelling patterns.

The research from [34] introduces a compositional technique for parametric model checking. This technique decomposes the underlying state transition graph of the analysed MC into strongly connected components (SCCs). Rational functions are then computed independently for each SCC and then combined to obtain the PMC result. In addition, [34] defines new polynomial factorisations that further improve the handling of the large expressions generated by PMC, and optimises the computation of the greatest common divisor used to simplify PMC rational expressions. The PMC technique from [34] is implemented by the recently released probabilistic model checker Storm [21], and achieves significant performance improvements over the previously developed PMC techniques. Like the technique from [34], ePMC is a compositional PMC method. However, while [34] operates with SCCs, the ePMC "components" are Markov chain fragments that may contain zero or more SCCs, or even parts of SCCs. This makes ePMC particularly flexible, and different from the technique from [34]. Further advantages of our method include the precomputation of the PMC expression associated with the Markov chain fragments, and the use of sets of formulae that include these PMC expressions without combining them. Finally, through using Storm in its second stage (cf. Section 5), ePMC leverages and extends the PMC technique from [34]. As shown in Section 8, this significantly improves the efficiency and scalability of parametric model checking.

One other characteristic that distinguishes ePMC from the techniques in [19], [32], [34] is its use of a domain-specific repository of precomputed QoS property expressions. As such, our ePMC method and pattern-aware probabilistic model checker do not offer the generality of the other techniques and model checkers. In return—for the domains for which such a repository has been built—ePMC can analyse parametric Markov chains up to several orders of magnitude faster, and yields much smaller and much more efficient to evaluate formulae than the current PMC approaches.

10 Conclusion

We presented ePMC, a tool-supported method for efficient parametric model checking. ePMC can efficiently analyse unbounded until and reachability reward PCTL formulae by precomputing closed-form expressions for the QoS properties of modelling patterns used frequently within a domain of interest. These expressions are then employed to considerably speed up the analysis of Markov chain models of systems from the same domain, and to generate sets of QoS property formulae that can be evaluated very efficiently. These improvements extend the applicability of parametric model checking to much larger models than previously feasible.

In our future work, we plan to extend the use of ePMC to further types of component-based systems. In particular, probabilistic model checking is increasingly used to analyse Markov chains comprising interchangeable modules within the important and broad domain of software product lines (e.g., [16], [28], [29], [41]). These interchangeable modules represent ideal ePMC modelling pattern candidates.

Furthermore, we envisage that the benefits of our work will extend to multiple applications of probabilistic and parametric model checking, and we plan to exploit ePMC in several of these applications. Thus, we intend to integrate ePMC with probabilistic model synthesis [26], [27], which is currently very computationally intensive due to the need to analyse numerous probabilistic model variants corresponding to different parameter values. Additionally, we plan to use ePMC instead of traditional parametric model checking in our recently introduced technique for formal verification with confidence intervals [10], [11], which can only analyse QoS properties defined by small to medium size closed-form expressions. Last but not least, we will build on our recent work from [8] to explore the use of ePMC within self-adaptive systems where not only the system parameters but also the system architecture needs to be reconfigured at runtime.

Acknowledgements

This work was partly funded by the Assuring Autonomy International Programme.

References

[1] J.-R. Abrial and S. Hallerstede, "Refinement, decomposition, and instantiation of discrete models: Application to event-b," *Fundamenta Informaticae*, vol. 77, no. 1-2, pp. 1–28, 2007.

[2] S. Andova, H. Hermanns, and J.-P. Katoen, "Discrete-time rewards model-checked," in *First International Workshop on Formal Modeling and Analysis of Timed Systems (FORMATS)*, 2004, pp. 88–104.

[3] C. Baier and J.-P. Katoen, *Principles of Model Checking*. MIT Press, 2008.

[4] E. Bartocci, R. Grosu, P. Katsaros, C. Ramakrishnan, and S. Smolka, "Model repair for probabilistic systems," in *17th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 2011, pp. 326–340.

[5] M. Benedikt, R. Lenhardt, and J. Worrell, "LTL model checking of interval Markov chains," in *19th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 2013, pp. 32–46.

[6] R. Calinescu, L. Grunske, M. Kwiatkowska, R. Mirandola, and G. Tamburrelli, "Dynamic QoS management and optimization in service-based systems," *IEEE Transactions on Software Engineering*, vol. 37, no. 3, pp. 387–409, May 2011.

[7] R. Calinescu, K. Johnson, and Y. Rafiq, "Developing self-verifying service-based systems," in *28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2013, pp. 734–737.

[8] R. Calinescu, D. Weyns, S. Gerasimou, M. U. Iftikhar, I. Habli, and T. Kelly, "Engineering trustworthy self-adaptive software with dynamic assurance cases," *IEEE Transactions on Software Engineering*, vol. 44, no. 11, pp. 1039–1069, 2018.

[9] R. Calinescu, M. Autili, J. Cámara, A. Di Marco, S. Gerasimou, P. Inverardi, A. Perucci, N. Jansen, J.-P. Katoen, M. Kwiatkowska *et al.*, "Synthesis and verification of self-aware computing systems," in *Self-Aware Computing Systems*. Springer, 2017, pp. 337–373.

[10] R. Calinescu, C. Ghezzi, K. Johnson, M. Pezzé, Y. Rafiq, and G. Tamburrelli, "Formal verification with confidence intervals to establish quality of service properties of software systems," *IEEE Transactions on Reliability*, vol. 65, no. 1, pp. 107–125, 2016.

[11] R. Calinescu, K. Johnson, and C. Paterson, "FACT: A probabilistic model checker for formal verification with confidence intervals," in *22nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 2016, pp. 540–546.

[12] ——, "Efficient parametric model checking using domain-specific modelling patterns," in *40th International Conference on Software Engineering: New Ideas and Emerging Results (ICSE:NIER)*, 2018, pp. 61–64.

[13] R. Calinescu, S. Kikuchi, and K. Johnson, "Compositional reverification of probabilistic safety properties for large-scale complex IT systems." in *17th Monterey Workshop: Large-Scale Complex IT Systems*, 2012, pp. 303–329.

[14] J. Cardoso, A. Sheth, J. Miller, J. Arnold, and K. Kochut, "Quality of service for workflows and web service processes," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 1, no. 3, pp. 281–308, 2004.

[15] T. Chen, E. M. Hahn, T. Han, M. Kwiatkowska, H. Qu, and L. Zhang, "Model repair for Markov decision processes," in *International Symposium on Theoretical Aspects of Software Engineering (TASE)*, 2013, pp. 85–92.

[16] P. Chrszon, C. Dubslaff, S. Klüppelholz, and C. Baier, "ProFeat: feature-oriented engineering for family-based probabilistic model checking," *Formal Aspects of Computing*, vol. 30, no. 1, pp. 45–75, 2018.

[17] F. Ciesinski and M. Größer, "On probabilistic computation tree logic," in *Validation of Stochastic Systems: A Guide to Current Research*, C. Baier, B. R. Haverkort, H. Hermanns, J.-P. Katoen, and M. Siegle, Eds. Springer, 2004, pp. 147–188.

[18] C. E. da Silva, J. D. S. da Silva, C. Paterson, and R. Calinescu, "Self-adaptive role-based access control for business processes," in *12th IEEE/ACM International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, 2017, pp. 193–203.

[19] C. Daws, "Symbolic and parametric model checking of discrete-time Markov chains," in *First International Conference on Theoretical Aspects of Computing (ICTAC)*, 2005, pp. 280–294.

[20] C. Dehnert, S. Junges, N. Jansen, F. Corzilius, M. Volk, H. Bruintjes, J.-P. Katoen, and E. Ábrahám, "PROPhESY: A PRObabilistic ParamEter SYnthesis Tool," in *27th International Conference on Computer Aided Verification (CAV)*, 2015, pp. 214–231.

[21] C. Dehnert, S. Junges, J.-P. Katoen, and M. Volk, "A Storm is coming: A modern probabilistic model checker," in *29th International Conference Computer Aided Verification (CAV)*, 2017, pp. 592–600.

[22] A. Filieri, C. Ghezzi, and G. Tamburrelli, "Run-time efficient probabilistic model checking," in *33rd International Conference on Software Engineering (ICSE)*, 2011, pp. 341–350.

[23] A. Filieri and G. Tamburrelli, "Probabilistic verification at runtime for self-adaptive systems." *Assurances for Self-Adaptive Systems*, vol. 7740, pp. 30–59, 2013.

[24] A. Filieri, G. Tamburrelli, and C. Ghezzi, "Supporting self-adaptation via quantitative verification and sensitivity analysis at run time," *IEEE Transactions on Software Engineering*, vol. 42, no. 1, pp. 75–99, 2016.

[25] S. Gallotti, C. Ghezzi, R. Mirandola, and G. Tamburrelli, "Quality prediction of service compositions through probabilistic model checking," in *4th International Conference on Quality of Software-Architectures (QoSA)*, 2008, pp. 119–134.

[26] S. Gerasimou, R. Calinescu, and G. Tamburrelli, "Synthesis of probabilistic models for quality-of-service software engineering," *Automated Software Engineering Journal*, vol. 25, no. 4, pp. 785–831, 2018.

[27] S. Gerasimou, G. Tamburrelli, and R. Calinescu, "Search-based synthesis of probabilistic models for quality-of-service software engineering," in *30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2015, pp. 319–330.

[28] C. Ghezzi and A. M. Sharifloo, "Verifying non-functional properties of software product lines: Towards an efficient approach using parametric model checking," in *15th International Conference on Software Product Lines SPLC*, 2011, pp. 170–174.

[29] ——, "Model-based verification of quantitative non-functional properties for software product lines," *Information & Software Technology*, vol. 55, no. 3, pp. 508–524, 2013.

[30] E. M. Hahn, T. Han, and L. Zhang, "Synthesis for pctl in parametric markov decision processes," in *NASA Formal Methods Symposium*. Springer, 2011, pp. 146–161.

[31] E. M. Hahn, H. Hermanns, B. Wachter, and L. Zhang, "PARAM: A model checker for parametric Markov models," in *22nd International Conference on Computer Aided Verification (CAV)*, 2010, pp. 660–664.

[32] E. M. Hahn, H. Hermanns, and L. Zhang, "Probabilistic reachability for parametric Markov models," *International Journal on Software Tools for Technology Transfer*, vol. 13, no. 1, pp. 3–19, 2011.

[33] H. Hansson and B. Jonsson, "A logic for reasoning about time and reliability," *Formal Aspects of Computing*, vol. 6, no. 5, pp. 512–535, 1994.

[34] N. Jansen, F. Corzilius, M. Volk, R. Wimmer, E. Ábrahám, J.-P. Katoen, and B. Becker, "Accelerating parametric probabilistic verification," in *11th International Conference on Quantitative Evaluation of Systems (QEST)*, 2014, pp. 404–420.

[35] K. Johnson, R. Calinescu, and S. Kikuchi, "An incremental verification framework for component-based software systems," in *16th ACM SIGSOFT Symposium on Component Based Software Engineering (CBSE)*, 2013, pp. 33–42.

[36] J.-P. Katoen, I. S. Zapreev, E. M. Hahn, H. Hermanns, and D. N. Jansen, "The ins and outs of the probabilistic model checker MRMC," *Performance Evaluation*, vol. 68, no. 2, pp. 90–104, 2011.

[37] J. G. Kemeny, J. L. Snell, and A. W. Knapp, *Denumerable Markov Chains, 2nd edition*, ser. Graduate Texts in Marhematics. Springer, 1976, vol. 40.

[38] 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)*, 2011, pp. 585–591.

[39] A. McIver and C. Morgan, *Abstraction, Refinement and Proof for Probabilistic Systems*, ser. Monographs in Computer Science. Springer, 2005.

[40] K. Sen, M. Viswanathan, and G. Agha, "Model-checking Markov chains in the presence of uncertainties," in *12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 2006, pp. 394–410.

[41] M. Ter Beek, A. Legay, A. L. Lafuente, and A. Vandin, "A framework for quantitative modeling and analysis of highly (re)configurable systems," *IEEE Transactions on Software Engineering (Early Access)*, 2018.

[42] L. Zeng, B. Benatallah, A. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "QoS-aware middleware for web services composition," *IEEE Transactions on Software Engineering*, vol. 30, no. 5, pp. 311–327, 2004.

Radu Calinescu is a Senior Lecturer within the Department of Computer Science at the University of York, UK. His main research interests are in formal methods for self-adaptive, autonomous, secure and dependable software, cyber-physical and AI systems, and in performance and reliability software engineering. He is an active promoter of formal methods at runtime as a way to improve the integrity and predictability of self-adaptive and autonomous software and cyber-physical systems and processes.

Colin Paterson is a Research Associate in the Assuring Autonomy International Programme at the University of York, where his research considers techniques for the verification of artificial intelligence. Colin recently completed a PhD which concerns the formal verification of operational processes using observation data to enhance the modelling of such processes and the accuracy of verification techniques.

Prior to this Colin obtained a PhD in control systems engineering in a collaboration with Jaguar Cars, before moving into industry where he designed bespoke web-based software solutions as well as a product suite for local government focused on governance, risk and compliance.

Kenneth Johnson is a Senior Lecturer in The School of Computer and Mathematical Sciences at Auckland University of Technology, New Zealand. He received his PhD in Computer Science from Swansea University, UK in 2007. He has held post-doctorate research positions at the University of York, Aston University and INRIA, Rennes. His research interests are formal modelling and verification of large-scale systems. Most recently, he has focused on automated model-based analysis of quality-of-service properties of systems at runtime. He is a member of the IEEE and serves on several program committees for international conferences featuring formal methods and cloud computing technology.