2017 IEEE/ACM 39th International Conference on Software Engineering

ProEva: Runtime Proactive Performance Evaluation Based on Continuous-Time Markov Chains

Guoxin Su*, Taolue Chen†, Yuan Feng‡, and David S. Rosenblum§
*School of Computing and Information Technology, University of Wollongong
†Department of Computer Science, Middlesex University London
‡Faculty of Engineering and Information Technology, University of Technology Sydney
§School of Computing, National University of Singapore

***Abstract*—Software systems, especially service-based software systems, need to guarantee runtime performance. If their performance is degraded, some reconfiguration countermeasures should be taken. However, there is usually some latency before the countermeasures take effect. It is thus important not only to monitor the current system status passively but also to predict its future performance proactively. Continuous-time Markov chains (CTMCs) are suitable models to analyze time-bounded performance metrics (e.g., how likely a performance degradation may occur within some future period). One challenge to harness CTMCs is the measurement of model parameters (i.e., transition rates) in CTMCs at runtime. As these parameters may be updated by the system or environment frequently, it is difficult for the model builder to provide precise parameter values. In this paper, we present a framework called ProEva, which extends the conventional technique of time-bounded CTMC model checking by admitting imprecise, interval-valued estimates for transition rates. The core method of ProEva computes asymptotic expressions and bounds for the imprecise model checking output. We also present an evaluation of accuracy and computational overhead for ProEva.**

***Keywords*-continuous-time Markov chain; imprecise parameters; performance; Quality-of-Service**

I. Introduction

Software systems need to guarantee runtime performance. If the system experiences downtime, some countermeasure has to be executed to reverse the performance. This performance-oriented reconfiguration is particularly important for service-based systems that must confirm to the Service-Level Agreement (SLA) between the provider and the client. One notable example of performance monitoring services is the Amazon Cloudwatch [1]. The majority of these services provide passive monitoring that relies on aggregating observable performance metrics. However, for runtime configurable systems some configuration latency may be caused by sever initialization, Virtual Machine (VM) installation and human actuation, among other reasons [2]. Therefore, besides passive monitoring, it is favorable to predict the change of those metrics proactively. When a requirement (as defined in the SLA) will likely be violated in some short time, some countermeasure must be triggered. This kind of *time-bounded metrics*, which involve computing a future likelihood, cannot be measured directly and are hard to simulate efficiently at runtime due to the presence of too many uncertain factors.

Continuous-time Markov chains (CTMCs) [3] are widely exploited formal models for performance analysis. CTMCs assume that basic system events (i.e., abstract state transitions) happen stochastically and with exponentially distributed delay. With mature algorithms and tools in the realm of *probabilistic model checking* [4], we can analyze a broad range of performance metrics, including the aforementioned time-bounded metrics. Owning to its analytic strength, probabilistic model checking has been utilized recently to reason about runtime software systems (e.g., to aid the engineering of self-adaptive software [5], [2], [6], [7]). However, in the runtime setting, one crucial challenge is *model building*, because the time and data for measuring the system parameters are restricted [8], and because those parameters may be updated frequently [9]. In case of a poor parameter measurement, the model checking output (i.e., the performance evaluation output) may be inaccurate. It is therefore advantageous to extend probabilistic model checking to admit a CTMC model with imprecisely measured parameters (e.g., interval-valued parameters). For example, if historical data show that a job arrival rate changes up to 5% in every hour, then this rate can be estimated as $\mu \cdot (1 \pm 5\%)$ within the next hour, where μ is the currently measured value. A countermeasure can then be determined based on the worst-case evaluation output of the metric.

Several methods in probabilistic model checking have been developed to deal with discrete-time Markov chains (DTMCs) with different forms of imprecise probability parameters [10], [11], [12]. It is well-known that in CTMCs, for computing metrics such as unbounded reachability and steady-state distributions, one only needs to consider the so called embedded DTMC of the CTMC. However, the model checking problem for CTMCs against time-bounded metrics, namely *time-bounded CTMC model checking* [13], requires different techniques. In a nutshell, it relies on the computation of *transient distributions* of CTMCs, which, in turn, usually employs the*uniformization* method [3]. The uniformization method can also be leveraged for sensitivity analysis for CTMCs (e.g., identifying performance-critical

1558-1225/17 $31.00 © 2017 IEEE
DOI 10.1109/ICSE.2017.51

Figure 1. Job Preprocessing in IaaS

parameters) [14], [15]. In this paper, we consider the following problem, which is more general than sensitivity analysis:

> *If the transition rates in a CTMC model are imprecise up to $\pm X\%$ then the time-bounded CTMC model checking output is affected by $\pm Y\%$.*

An inference of Y from X can enhance the applicability of probabilistic model checking to runtime performance evaluation. More specifically, the CTMC model builder need not to provide precise parameter values in order to achieve reliable runtime proactive evaluation of time-bounded metrics.

We present ProEva, a framework of Proactive performance Evaluation to address the above-mentioned analysis problem. ProEva contains two computational methods on top of time-bounded CTMC model checking, First, it extends the use of the uniformization method to compute an *asymptotic expression* for the transition distribution of a parametric CTMC. This generalizes the method reported in the sensitivity analysis literature [14], [15]. Second, by specifying imprecise parameters as intervals, it exploits this asymptotic expression to compute *asymptotic bounds* for the evaluation output. We also present the tool support for ProEva and an empirical evaluation for its computational overhead and accuracy. This technique of ProEva extends our previous work on DTMCs [16] to the setting of time-bounded CTMC model checking.

The remainder of the paper is organized as follows. Section II presents the performance modeling with a concrete example, and a high-level description of ProEva. Section III presents the basic technique of time-bounded CTMC model checking. Section IV presents the core method of ProEva. Section V presents the tool support and a case study. Section VI reports the related work. Section VII concludes the paper.

II. Performance Modeling and Framework

In this section, we present the CTMC-based performance modeling with a concrete example, and highlight the motivation and significance of ProEva.

A. Running Example

In an Infrastructure-as-a-Service (IaaS) cloud, to achieve the cost-effective usage of hardware resources (e.g., CPU, RAM and memory), the (physical) servers can be grouped into different pools, e.g., a running pool and a sleeping pool. More running servers enable more deployed Virtual Machines (VMs) to serve clients, but also cause higher maintenance cost (e.g., power consumption). Server management aims to guarantee the Quality-of-Service (QoS) of the IaaS cloud when optimizing the maintenance cost.

```
module Queues
  HQ: [0,N];
  LQ: [0,N]; //N the max queue size//
  // HQ (LQ) the high (low) priority queue//
  [enqueue1] HQ < N -> HQ' = HQ + 1;
  [enqueue2] LQ < N -> LQ' = LQ +1;
  [dequeue] HQ > 0 -> HQ' = HQ - 1;
  [dequeue] HQ = 0 & LQ > 0 -> LQ' = LQ - 1;
endmodule
```

Figure 2. Specification of Queues

```
module Receiver
  s: [1,1];
  [enqueue1] mu1: true -> s' = s;
  [enqueue2] mu2: true -> s' = s;
  // mu1, mu2 are the arrival rates.//
endmodule
module Assigner
  r: [1,1];
  [dequeue] lambda: true -> r' = r;
  // lambda is the exit rate.//
endmodule
```

Figure 3. Specification of Receiver and Assigner

One important QoS indicator is the *service provisioning delay*. Suppose each job passes through three phases before being processed by the VMs (as shown in Figure 1). New jobs first enter a Receiver and are sorted into two Queues with high and low priorities. These jobs are then handled by the Assigner in order. In particular, low-priority jobs are handled only when no high-priority job is awaiting service. Once a job exits from a VM, the Assigner is notified and another job is assigned to this vacant VM. The Receiver, Queues and Assigner constitute a component (called Preprocessor) of the IaaS cloud system.

Since the provisioning delay is mostly due to queueing, we measure it as queue lengths. For better explanation, Figure 2 specifies the Queues as a module in the PRISM-style modeling language [17]. The module contains four update rules labeled with one of the three actions `enqueue1`, `enqueue2` and `dequeue`. Suppose we require that the high-priority and low-priority queues contain no more than N_1 and N_2 numbers of jobs, respectively, where $N_1, N_2 \leq N$. If such a requirement is violated, then one or more sleeping servers should be awakened to work. With more working VMs, the job exit rate or, equivalently, the speed of dequeueing the two queues is improved. But before an awaken server becomes effective, the instantiation of a VM from a pre-installed image takes time. Therefore, in order to avoid the requirement violation, it is useful to know the likelihood that this requirement will be violated within t time units, where t may be the time required for VM instantiation. But different from the queue lengths (i.e., `HQ` and `LQ`), such a time-bounded metric cannot be measured directly. Moreover, even though it might be simulated, an extensive simulation is infeasible at runtime due to the time and data constraints.

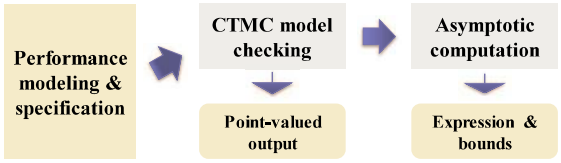


Figure 4. Overview of Framework ProEva

B. CTMC Model Building

To analyze the above-mentioned time-bounded metric for the IaaS Preprocessor, we adopt a CTMC-based analytic approach. CTMCs are widely utilized for performance and reliability analysis of cloud systems. To use this approach, we specify the Receiver and Assigner in Figure 3. The update rules therein are synchronized with the labeled with the same action in the modules Queues and triggered with the rates μ_1, μ_2 and λ. Both modules contain one (abstract) state only, even though their implementation may be complex in a real IaaS system (for example, the Receiver is responsible for blocking invalid or duplicate requests, and the Assigner implements a sophisticated resource-provisioning algorithm). Clearly, the semantics of the three modules is one CTMC model. With probabilistic model checking, the above-mentioned time-bounded metrics can be formally analyzed. In particular, the likelihood of requirement violation is computed as a probability. The underlying technique of probabilistic model checking will be presented later in Section III; for now, we focus on the model building.

The Receiver and Assigner are specified at a high level of abstraction in order to simplify the model while fulfilling our analysis purpose. Besides abstraction, we also implicitly make the following two model building assumptions:

- The time intervals of job arrivals and exits are exponentially distributed and mutually independent;
- The average lengths of those time intervals (i.e., $1/\mu_1$, $1/\mu_2$ and $1/\lambda$) are fixed.

At least, we need to make these two assumptions for the IaaS system along t time units from the present (where t is specified in the time-bounded metric under consideration). The first assumption enables an analytic method for CTMCs based on the mathematical properties of exponential distributions. The second assumption, however, imposes a burden on model building. Namely, even if the *present* values of the three parameters can be determined effectively, their *future* values may be different as the environment changes and thus are difficult to elicit precisely. Therefore, if these measured values inputted to the tool are imprecise in a nontrivial scale, the performance evaluation output is likely to be unreliable, which may lead to an incorrect countermeasure.

C. ProEva Highlight and Significance

To overcome the aforementioned problem of parameter measurement in CTMC model building, not only does our framework ProEva produces a concrete evaluation output, but it also addresses the effect of imprecise parameters on this output. As illustrated in Figure 4, ProEva contains three stages: (i) performance modeling and specification, (ii) time-bounded CTMC model checking and (iii) the core method of *asymptotic computation*. We have explained performance modeling and specification in the previous subsection. Before delving into the technical details of the last two stages, we highlight our core method and its practical significance.

Suppose we employ time-bounded CTMC model checking to analyze the time-bounded metrics of the IaaS Preprocessor and compute a (point-valued) evaluation output, say, p_0, which refers to the likelihood that the queue lengths requirement is violated within t time. As μ_1, μ_2 and λ may be imprecise, ProEva admits the interval estimations in the form of $\mu_1(1 \pm a_1)$, $\mu_2(1 \pm a_2)$ and $\lambda(1 \pm a_3)$, respectively, for some a_i with $i \in \{1, 2, 3\}$. Notice that ProEva does not assume a particular technique of interval estimation. In practice, one or more copies of a_i may be inferred from the historical values of the three parameters that were measured before. For example, the (time-stamped) historical values of the three rates may demonstrate that, with a confidence level $c \cdot 100\%$, all of them change up to $a_c \cdot 100\%$ in every t time units, where say, $c \in \{0.95, 0.99\}$. As mentioned in Section I, our focus is on the inference of $p_0(1 \pm b)$ for some b, which represents the range of the imprecise output, from the given values of a_1, a_2 and a_3. More specifically, our core method comprises three steps. First, we parameterize the three rates in the form $\mu_1 \cdot (1 + x_1)$, $\mu_2 \cdot (1 + x_2)$ and $\lambda \cdot (1 + x_3)$, where x_1, x_2 and x_3 are variables. As a result, the evaluation output becomes a function $f(x_1, x_2, x_3)$ with $p_0 = f(0, 0, 0)$. We semi-formally (for now) present the *asymptotic expression* of f as the following quadratic fragment of its Taylor series:

$$f(x_1, x_2, x_3) \approx p_0 + f_1(x_1, x_2, x_3) + f_2(x_1, x_2, x_3)$$

where f_1 (resp. f_2) is a linear (resp. homogenous quadratic) function. Second, we compute the functions f_1 and f_2 by extending the standard uniformization method for computing p_0. Finally, given a_1, a_2 and a_3, we compute a pair of *asymptotic bounds* $p_0 + b_*$ and $p_0 + b^*$ based on f_1 and f_2, which estimate the worst-case value of f. Slightly modifying the two bounds, we get $p_0(1 \pm b)$ where $b = \max\{-b_*, b^*\} \cdot p_0^{-1}$.

As a result, ProEva extends the flexibility of time-bounded CTMC model checking by relaxing its typically strict expectation for parameter estimation in model building. In the IaaS Preprocessor example, without resorting to the point-valued rates, which may be unrealistic to obtain in model building, we can determine a countermeasure just based on the worst-case likelihood of requirement violation. While time-bounded CTMC model checking is well investigated and applied, clearly the practicality of ProEva is also relevant to the performance of the asymptotic computation. We emphasize that the most important reason for our pursuit of asymptotic bounds is the complexity of computing a

closed-form expression or *exact bounds*, which are defined later in Sections IV-A. The philosophy behind ProEva is an adequate trade-off between *accuracy* and *computational overhead*. Theoretical analysis and empirical evaluation of the two aspects are presented later in Sections IV and V, respectively. But briefly, although small-scale parameter imprecision may cause significant imprecision on the evaluation output, ProEva can produce reasonably accurate bounds for the imprecise output. Also, the time of asymptotic computation in ProEva increases roughly quadratically against the number of parameters and, when the number of parameters is fixed, is proportional to standard time-bounded CTMC model checking. These two characteristics demonstrate the practicality of ProEva.

III. Time-Bounded CTMC Model Checking

In this section, we recall the background of time-bounded CTMC model checking. Our presentation is tailored for the purpose of this paper; for a comprehensive reference on this technique, the readers are referred to Baier et al. [18].

A. Markov Chain and Temporal Property

Definition 1 (Continuous-Time Markov Chain)**.** *A CTMC is a tuple $\mathcal{M} = (S, \alpha, \mathbf{R}, AP, L)$ where*

- *S is a state space with $|S| = n > 0$,*
- *α is an initial distribution over S,*
- *$\mathbf{R} : S \times S \to \mathbb{R}_{\geq 0}$ is a* (transition) rate matrix*,*
- *AP is a set of atomic propositions, and*
- *$L : S \to 2^{AP}$ assigns atomic propositions to each state.*

It is also natural to represent a rate matrix as a directed graph (i.e., digraph). The CTMC model of IaaS Preprocessor (c.f., Figures 2 and 3) is depicted in Figure 5a, where—for easy exposition—we let $N_1 = 1$ and $N_2 = N = 2$. Let $\mathbf{E}$ be a diagonal matrix such that $\mathbf{E}[i,i] = \sum_{j=1}^{n} \mathbf{R}[i,j]$ for each $1 \leq i \leq n$. Then, $\mathbf{Q} = \mathbf{R} - \mathbf{E}$ is the *infinitesimal generator* of $\mathcal{M}$. The behavior of a CTMC can be characterized by the following ordinary differential equation (ODE) system [19]:

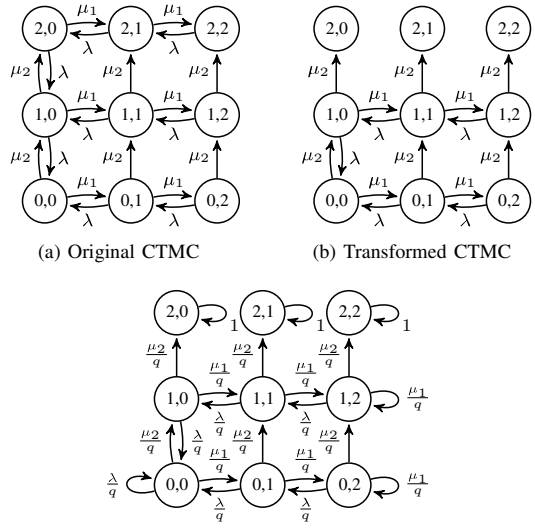
$$\frac{d}{dt}\pi(t) = \pi(t) \cdot \mathbf{Q}, \quad \text{given } \pi(0) = \alpha. \tag{1}$$

In the literature, $\pi(t)$ is called the *transient distribution* over S at time t, which plays an important role in time-bounded CTMC model checking. Let $\pi_s(t)$ denote the probability that $\pi(t)$ assigns to state $s \in S$. Intuitively, $\pi_s(t)$ is the probability of staying at s at time t.

Although we mainly work on CTMCs, we also present the model of discrete-time Markov chains (DTMCs) for comparison and illustrative purposes.

Definition 2 (Discrete-Time Markov Chain)**.** *A DTMC is a tuple $\mathcal{D} = (S, \alpha, \mathbf{P}, AP, L)$ where*

- *$\mathbf{P} : S \times S \to [0,1]$ is a* transition probability matrix *such that $\sum_{j=1}^{n} \mathbf{P}[i,j] = 1$ for all $1 \leq i \leq n$, and*
- *all other components are defined as in $\mathcal{M}$ (cf. Def. 1).*



(a) Original CTMC (b) Transformed CTMC

(c) Uniformized DTMC

Figure 5. Illustrative Models of IaaS Preprocessor

The time-bounded performance metrics that we investigate in this paper are formalized as *time-bounded temporal properties*, which are one fragment of the Continuous Stochastic Logic (CSL) [18]. Despite a prominent logic for CTMC model checking, the full-fledged syntax of CSL leads to complications that are unnecessary for our purposes.

Definition 3 (Time-Bounded Temporal Property)**.** *Let ϕ, ψ be propositional logic formulae over AP. Time-bounded temporal properties are of the form $\psi \mathcal{U}^{\leq t} \phi$.*

Informally, $\mathcal{U}$ is a "time-bounded until" connective, and $\psi \mathcal{U}^{\leq t} \phi$ expresses the property "ψ is true until ϕ is true within t time". The formal semantics of $\psi \mathcal{U}^{\leq t} \phi$ requires the definition of *paths* and the *probability Borel measures* for CTMCs [18]. Due to space limitations, we simply mention that $\mathbf{Pr}(\mathcal{M} \vDash \psi \mathcal{U}^{\leq t} \phi)$ is *the probability of $\mathcal{M}$ satisfying $\psi \mathcal{U}^{\leq t} \phi$*. Let tt be a tautology and $\Diamond^{\leq t} \phi$ abbreviate $tt \mathcal{U}^{\leq t} \phi$. For example, the formula $\Diamond^{\leq t} \phi_0$ with $\phi_0 = \neg(\texttt{HQ} \leq N_1 \wedge \texttt{LQ} \leq N_2)$ expresses the time-bounded metric of queues specified in Section II-A. Recall that `HQ` (resp., `LQ`) is the length of the high-priority (resp., low-priority) queue.

Now the model checking problem for CTMCs against time-bounded temporal properties, or *time-bounded CTMC model checking*, is defined as follows:

Input: model $\mathcal{M}$ and property $\psi \mathcal{U}^{\leq t} \phi$,
Output: probability $\mathbf{Pr}(\mathcal{M} \vDash \psi \mathcal{U}^{\leq t} \phi)$.

B. Model Transformation and Uniformization

To solve the time-bounded model checking problem, we resort to the *transient analysis* of CTMCs. A state s of $\mathcal{M}$ is *absorbing* if $\mathbf{R}[s, s'] = 0$ for all $s' \in S$. We define a CTMC $\mathcal{M}[\phi]$ by making the states in $\mathcal{M}$ that satisfy ϕ as

absorbing states. Formally, let $\mathbf{R}^{[\phi]}$ denote the rate matrix of $\mathcal{M}[\phi]$; for all $s, s' \in S$, if $s \vDash \phi$ then $\mathbf{R}^{[\phi]}[s, s'] = 0$, and if $s \nvDash \phi$ then $\mathbf{R}^{[\phi]}[s, s'] = \mathbf{R}[s, s']$.

Proposition 1 (CTMC Transformation [18])**.** *Given $\mathcal{M}$ and $\psi \mathcal{U}^{\leq t} \phi$, let $\pi(t)$ be a transient distribution of $\mathcal{M}[\phi \vee \neg\psi]$ and $S' = \{s \in S \mid s \vDash \phi\}$. It holds that*

$$\mathbf{Pr}(\mathcal{M} \vDash \psi \mathcal{U}^{\leq t} \phi) = \sum_{s \in S'} \pi_s(t). \tag{2}$$

The CTMC in Figure 5b is transformed from the one in Figure 5a w.r.t. $\Diamond^{\leq t} \phi_0$. States $(2, j)$ with $j = 0, 1, 2$ are absorbing states in the transformed model. By Proposition 1, the probability that $\Diamond^{\leq t} \phi_0$ is satisfied by the model in Figure 5a equals to the transient probability of locating at the absorbing states of the model in Figure 5b at time t.

Owing to Proposition 1, it suffices to work on transient distributions. In probabilistic model checking, the state-of-the-art method to compute transient distributions is a numerical method based on *uniformization*. This method is widely adopted owing to its scalability and advantage in numerical computation, although other methods do exist [3]. The first step of this method is to choose some $q \geq |\mathbf{Q}[i, j]|$ for all $1 \leq i, j \leq n$. Let $\mathbf{U} = \mathbf{Q}/q + \mathbf{I}$. It is easy to verify that $\mathbf{U}$ is a transition probability matrix, and thus we obtain a specific DTMC. As an example, the DTMC in Figure 5c where $q = \mu_1 + \mu_2 + \lambda$ is uniformized from the transformed CTMC in Figure 5b. The following proposition is a well-known key technique for the uniformization method.

Proposition 2 (Uniformization [20])**.** *It holds that*

$$\pi(t) = \sum_{r=0}^{\infty} e^{-qt} \frac{(qt)^r}{r!} (\alpha \cdot \mathbf{U}^r). \tag{3}$$

Since the right-hand side of Equation (3) is an infinite series, a suitable truncation has to be made in the actual computation of π.[1] More specifically, fixing a small number $\epsilon > 0$, there is an (usually large) number r_ϵ such that $\sum_{r=0}^{r_\epsilon} e^{-qt} \frac{(qt)^r}{r!} > 1 - \epsilon$. The absolute difference between π and the sum of the first $r_\epsilon + 1$ terms in the series is bounded by ϵ. This holds because α is a probability distribution and $\mathbf{U}$ is a transition probability matrix (in which all entries of $\mathbf{U}$ are non-negative and the sum of each row equals 1). By contrast, $\mathbf{Q}$ may contain negative and large entries, making an error bound difficult to achieve. Further treatment on this truncation problem can be found in the literature [3].

C. Other Performance Metrics

We have restricted our attention to time-bounded temporal properties. Other performance metrics include time-unbounded temporal properties and state-steady metrics, which can be expressed in the full-fledged syntax of CSL. For these performance metrics, the problem of CTMC model checking can be reduced to DTMC model checking. This should not be surprising—as the time-unboundedness and steady-state make the delay of transitions insignificant, a CTMC can be reduced to its "embedded" DTMC. The computation underlying DTMC model checking is different from, and in general more efficient than, the uniformization method. For a tutorial on DTMC model checking, readers can refer to the standard literature [21, Chapter 10].

[1]Notice that the truncation used here is different from the one due to the finite representation of real numbers.

IV. Asymptotic Computation

In this section, we present the core method of ProEva, including the CTMC parameterization and the computation of asymptotic expressions and asymptotic bounds.

A. Parameterization, Expression and Bounds

The purpose of parameterizing a CTMC is to represent the uncertainty of the transition rates. We use a vector variable $\vec{x} = (x_1, \ldots, x_m)$ to associate with the transition rates in a CTMC system model that may be updated at runtime. A *parameterization* of the transition matrix $\mathbf{R}$, denoted $\mathbf{R}(\vec{x})$, is defined as follows: For all $1 \leq i, j \leq n$,

- If $\mathbf{R}[i, j] = 0$ then $\mathbf{R}(\vec{x})[i, j] = 0$, and
- If $\mathbf{R}[i, j] = p > 0$ then either $\mathbf{R}(\vec{x})[i, j] = p$ or $\mathbf{R}(\vec{x})[i, j] = p + x_k$ for some $1 \leq k \leq m$.

The first condition prohibits any structural change of the original rate matrix $\mathbf{R}$ (when viewed as a digraph). In other words, we assume that only the quantitative characteristic rather than the structure of a CTMC is subject to imprecision. $\mathbf{Q}(\vec{x})$ and $\mathbf{U}(\vec{x})$, the parameterizations of $\mathbf{Q}$ and $\mathbf{U}$ respectively, are defined accordingly. Wherever $\vec{x}$ occurs, we always assume $\vec{x} \in \vec{D} = D_1 \times \ldots \times D_m$ such that $\mathbf{R}(\vec{x})$ is a non-negative matrix for any $\vec{x} \in \vec{D}$.

Definition 4 (Parametric CTMC)**.** *Given a CTMC $\mathcal{M} = (S, \alpha, \mathbf{R}, AP, L)$, a parametric CTMC is a tuple $\mathcal{M}(\vec{x}) = (S, \alpha, \mathbf{R}(\vec{x}), AP, L)$ where $\mathbf{R}(\vec{x})$ is a parameterization of the rate matrix $\mathbf{R}$.*

The transient distribution of $\mathcal{M}$ at time t is parametric, denoted $\pi(t, \vec{x})$. By Proposition 1, to deal with time-bounded temporal properties, we only need to consider the *transient measure* of $\mathcal{M}(\vec{x})$: Given $t \geq 0, S' \subseteq S$, let

$$\mathit{\Pi}(\vec{x}) = \sum_{s \in S'} \pi_s(t, \vec{x}) = \pi(t, \vec{x}) \cdot \mathbf{1}_{S'}, \tag{4}$$

where $\mathbf{1}_{S'}$ is a column vector with 1 in all S'-entries and 0 in all other entries. From Proposition 2, it is easy to see that $\mathit{\Pi}(\vec{x})$ is smooth (i.e., infinitely differentiable) and thus its Taylor expansion at $\vec{x} = 0$ exists.

We have provided two alternative characterizations for $\mathit{\Pi}(\vec{x})$ based on the ODE (i.e., Equation (1)) and the (infinite) series (i.e., Equation (3)) of the (parametric) transient distribution. For runtime evaluation, as parameters $\vec{x}$ may be updated frequently, it seems advantageous to obtain a

(finite) *closed-form expression* of $\Pi(\vec{x})$ whose value can be efficiently calculated once $\vec{x}$ is instantiated. However, except for a highly structured CTMC, the computation of such an expression is very costly and impractical.

To characterize the imprecise model checking output, an *idealized* characterization is *exact bounds*. An exact upper bound of $\Pi(\vec{x})$ is the optimal value of the problem:

$$\text{Maximize } \Pi(\vec{x}) \quad \text{subject to } \vec{x} \in \vec{I}, \tag{5}$$

where $\vec{I} = I_1 \times \ldots \times I_m \subseteq \vec{D}$ with each I_i being an interval $0 \in \vec{I}_i$. Notice that the maximum exists as $\vec{I}$ is compact and $\Pi(\vec{x})$ is continuous. Intuitively, $\vec{I}$ may be a "confidence region" inferred from the historical data of $\vec{x}$. The exact lower bound of $\Pi(\vec{x})$ is the optimal value of the minimization counterpart of Problem (5). Unfortunately, these exact bounds are also very hard to compute.

B. Asymptotic Expression

In view of the computational bottleneck, we pursue approximated solutions of Problem (5) and its minimization counterpart. In this subsection, we present a numerical approach to compute an asymptotic expression for a given transient distribution. In the next subsection, we will utilize this expression to derive a pair of asymptotic bounds.

Some existing work [14], [15] already extended the uniformization method to compute the partial derivatives for transient distributions of (parametric) CTMCs. Their main goal is to perform sensitivity analysis, such as identifying the most critical transition rates for transient performance. Our goal, however, is to estimate the effect of imprecise transition rates on transient measures (and thus on time-bounded performance metrics). To achieve higher accuracy with a reasonable computational overhead, we further extend the uniformization method to compute the second-order partial derivatives for transient measures.

To present the computation, we first give an auxiliary definition. Let $\mathbf{X}$ be the parametric matrix such that $\mathbf{U}+\mathbf{X} = \mathbf{U}(\vec{x})$. For each $1 \le i \le m$, we define $\mathbf{K}_i$ to be the matrix obtained from $\mathbf{X}$ by replacing x_i with 1, and variables x_j with 0 for all $1 \le j \le m$ and $j \ne i$. By adopting a matrix form of partial derivatives, we have that $\mathbf{K}_i = \partial\mathbf{U}(\vec{x})/\partial x_i$.

The computation consists of three iterations that involve matrix-vector multiplication, corresponding to the constant, linear, and (homogeneous) quadratic parts of $\Pi(\vec{x})$, respectively. The first iteration is as follows.

$$\begin{aligned} \Gamma^{(0)} &= \alpha, \\ \Gamma^{(r)} &= \Gamma^{(r-1)} \cdot \mathbf{U}, \text{ if } r \le r_\epsilon. \end{aligned} \tag{6}$$

Recall that r_ϵ is a truncation length to guarantee a given error bound ϵ in the computation of π. In fact, this iteration is a part of the computation of transient distributions based on the (standard) uniformization method (cf. Section III-B). $\Gamma^{(r)}$ refers to the vector generated in the r-th iteration. When $r \ge 1$, $\Gamma^{(r)}$ is only based on $\Gamma^{(r-1)}$ for each r.

The second iteration is as follows: For each $1 \le i \le m$,

$$\begin{aligned} \Delta_i^{(0)} &= \alpha \cdot \mathbf{K}_i, \\ \Delta_i^{(r)} &= \Delta_i^{(r-1)} \cdot \mathbf{U} + \Gamma^{(r)} \cdot \mathbf{K}_i, \text{ if } r \le r_\epsilon. \end{aligned} \tag{7}$$

In this iteration, $\Delta_i^{(r)}$ is based on $\Delta_i^{(r-1)}$ and $\Gamma^{(r)}$ for each $r \ge 0$.

The last iteration is as follows: For each $1 \le i, j \le m$,

$$\begin{aligned} \Theta_{i,j}^{(0)} &= \alpha \cdot \mathbf{K}_i \cdot \mathbf{K}_j, \\ \Theta_{i,j}^{(r)} &= \Theta_{i,j}^{(r-1)} \cdot \mathbf{U} + \Delta_i^{(r)} \cdot \mathbf{K}_j, \text{ if } r \le r_\epsilon. \end{aligned} \tag{8}$$

Similarly, in this iteration, $\Theta_{i,j}^{(r)}$ is based on $\Theta_{i,j}^{(r-1)}$ and $\Delta_i^{(r)}$ for each $r \ge 1$. Obviously, the three iterations are sequentially dependent. As a result, although we have separated the presentation of Iterations (6) to (8) for readability, we can combine them in a single iteration for implementation to avoid duplicated computation,.

The following theorem, which is the main technical result of this paper, bridges our iterative computation and the quadratic expression in the Taylor series of $\Pi(\vec{x})$.

Proposition 3. *Suppose* $r_\epsilon = \infty$. *For all* $1 \le i, j \le m$,

$$\sum_{r=0}^{\infty} e^{-qt}\frac{(qt)^r}{r!} \cdot \Gamma^{(r)} \cdot \mathbf{1}_{S'} = \Pi(0), \tag{9}$$

$$\sum_{r=1}^{\infty} e^{-qt}\frac{(qt)^r}{r!} \cdot \Delta_i^{(r-1)} \cdot \mathbf{1}_{S'} = \frac{\partial\Pi(0)}{\partial x_i}, \tag{10}$$

$$\sum_{r=2}^{\infty} e^{-qt}\frac{(qt)^r}{r!} \cdot (\Theta_{i,j}^{(r-2)} + \Theta_{j,i}^{(r-2)}) \cdot \mathbf{1}_{S'} = \frac{\partial\Pi(0)}{\partial x_i \partial x_j}. \tag{11}$$

Due to space limitations, a formal proof of Proposition 3 is given in the appendix of the technical report [22]. Notice that Equation (9) is corresponding to Proposition 2; we present it here for a comparison to Equations (10) and (11). Again, the idea of Equation (10) has been exploited in [14] and also reported in [15] for sensitivity analysis of transient distributions. But we show that the uniformization method can be further utilized to deal with the second-order partial derivatives. Based on Proposition 3, the (quadratic) asymptotic expressions can be inferred immediately. Formally, let $c_i = \partial\Pi(0)/\partial x_i$ and $c_{i,j} = \partial\Pi(0)/\partial x_i \partial x_j$. According to Taylor's Theorem, we have

$$\Pi(\vec{x}) = \Pi(0) + \Pi_1(\vec{x}) + \Pi_2(\vec{x}) + o(\|\vec{x}\|^2), \tag{12}$$

where $\Pi_1(\vec{x}) = \sum_{i=1}^m c_i x_i$, $\Pi_2(\vec{x}) = \sum_{i,j=1}^m \frac{1}{2} c_{i,j} x_i x_j$ and the small "o" notation $o(\|\vec{x}\|^2)$ refers to a polynomial of degree greater than 2 over $\vec{x}$.

We now consider the complexity of the iterative computation. In particular, we analyze the computational overhead with respect to the standard uniformization method, namely, the computation of $\Pi(0)$. In our setting, each matrix-vector multiplication involves a matrix of size $n \times n$ and a vector of size $n = |S|$, hence the operation takes at most n^2 time.

(Notice however that this estimation is rather pessimistic, as matrices $\mathbf{K}_i$'s are indeed sparse, so the computation can be speeded up greatly.) Iteration (6) requires r_ϵ matrix-vector multiplications, while Iterations (7) and (8) require $2r_\epsilon + 1$ and $2r_\epsilon + 2$ matrix-vector multiplications, respectively. In general, we have that $r_\epsilon \leq e^2 qt + \log(\frac{1}{\epsilon})$, where $e \approx 2.718$ is the base of the natural logarithm [13]. Since there are at most m variables in the model, the number of matrix-vector multiplications for $\mathit{\Pi}_1(\vec{x})$ is $m(2r_\epsilon + 1)$ and that for $\mathit{\Pi}_2(\vec{x})$ is $m^2(2r_\epsilon + 2)$. If m is not very large, the total computational overhead of the quadratic expression computation is acceptable. We mention that our method can be generalized even further to compute the higher-order fragment of the Taylor series of $\mathit{\Pi}(\vec{x})$ but at the price of higher computational overhead.

C. Asymptotic Bounds

In this subsection, we present a flexible procedure to compute approximated optimal values for Problem (5) and its minimization counterpart based on the asymptotic expression $\mathit{\Pi}(0) + \mathit{\Pi}_1(\vec{x}) + \mathit{\Pi}_2(\vec{x})$ of $\mathit{\Pi}(\vec{x})$.

Replacing the objective function $\mathit{\Pi}(\vec{x})$ in Problem (5) with $\mathit{\Pi}_1(\vec{x}) + \mathit{\Pi}_2(\vec{x})$, we obtain a Quadratic Program (QP)

$$\text{Maximize } \mathit{\Pi}_1(\vec{x}) + \mathit{\Pi}_2(\vec{x}) \quad \text{subject to } \vec{x} \in \vec{I}. \tag{13}$$

Although the QP problem is NP-complete in general, practical algorithms (e.g., by LU factorization) exist and perform well for a medium-sized problem with mature tool support. However, if Problem (13) is of large scale (e.g., there are many variables in the objective function $\mathit{\Pi}_1(\vec{x}) + \mathit{\Pi}_2(\vec{x})$), or if we need to solve Problem (13) for a large number of different "confidence regions" $\vec{I}$'s, we may further relax the Problem (13) in the following steps. First, we compute an optimal solution of a simple Linear Program (LP):

$$\text{Maximize } \mathit{\Pi}_1(\vec{x}) \quad \text{subject to } \vec{x} \in \vec{I}. \tag{14}$$

Notice that, since each I_i in $\vec{I}$ is an interval independent of others, an optimal solution, say, $\vec{v}$ of Problem (14) can be computed much more efficiently than a general LP problem. Specifically, we just need to know whether the coefficient of x_i in $\mathit{\Pi}_1(\vec{x})$ is negative or non-negative, and let v_i be the left (resp., right) end point of I_i if it is negative (resp., non-negative). Then, we compute $\mathit{\Pi}_1(\vec{v}) + \mathit{\Pi}_2(\vec{v})$ as an approximated optimal value of Problem (13).

A pair of asymptotic bounds b^* and b_* are the values obtained by—directly or approximately—solving Problem (13) and its minimization counterpart, respectively. In practice, we can rewrite (and slightly modify) the asymptotic bounds as $\mathit{\Pi}(0)(1 \pm b)$ where $b = \max\{-b_*, b^*\} \cdot \mathit{\Pi}(0)^{-1}$.

V. Tool Support and Evaluation

In this section, we use the running example IaaS Preprocessor to illustrate the tool support, and a system taken from the probabilistic model checking benchmarks for the empirical evaluation.

```
ctmc IaaS_Preprocessor
  ... // The PRSIM specification
        of the three modules goes here.//
label "phi" = HQ > N1 or LQ > N2;
  //define "phi" as "HQ > N1 or LQ > N2"//
label "psi" = true;
rewards "enqueue1"
  [enqueue1] true: r1;
  //assign reward r1>0 to enqueue1 action//
endrewards
rewards "enqueue2"
  [enqueue2] true: r2;  //r2>0//
endrewards
rewards "dequeue"
  [dequeue] true: r3; //r3>0//
endrewards
```

Figure 6. Complete Specification of IaaS Preprocessor

A. Tool Support

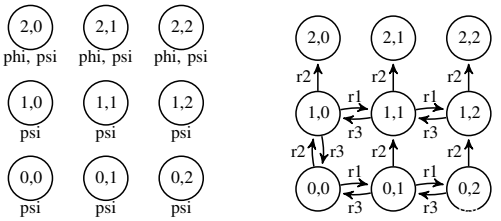
All three stages of ProEva (cf. Figure 4) are tool-supported. Performance modeling and specification of ProEva employs the PRISM model checker. Time-bounded CTMC model checking and asymptotic computation of ProEva, which manipulate (among others) vectors and matrices exported from PRISM, are implemented in Matlab.

We use the IaaS Preprocessor to illustrate the tool support. The modules Queues, Receiver and Assigner are specified in Figures 2 and 3. Technically, the specification of Receiver and Assigner does not follow the syntax of PRISM modeling language, which is not important for illustrative purposes. A complete specification of the IaaS Preprocessor is in Figure 6, which contains the three modules with labels and reward structures. The two labels encode a time-bounded property of the form $\psi \mathcal{U}^{\leq t} \phi$, which equals $\Diamond^{\leq t} \phi$ as ψ is declared as a tautology, for some (unspecified) t. The two reward structures are for parameterizing the rates μ_1, μ_2 and λ in the actions. The data structures exported from the specification in Figure 6 (using the PRISM inbuilt model export function) encode a uniformized and parametric DTMC model for the IaaS Preprocessor and a time-bounded metric. For example, the exported data structures can be represented in Figures 7a and 7b, which together yield a parametric version of the DTMC in Figure 5c where $N_1 = 1$ and $N_2 = N = 2$.

Our prototype implementation consists of three components. The first component "M_tran" reads the exported data structures and implements the model transformation. The second component "E_comp" implements the uniformization method and the core method of ProEva, namely, the asymptotic computation. The last component "Val" is a function that returns an asymptotic bound whenever the imprecision intervals of the parameters are specified.

B. Research Questions and Evaluation Methodology

To evaluate the accuracy and computational overhead of ProEva, we consider three research questions (RQs):



(a) Labeled States (b) Rewarded Transitions

Figure 7. Exported Data Structures

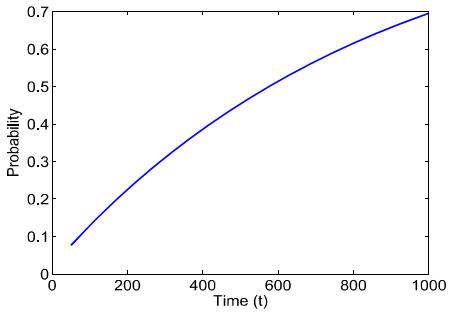


Figure 8. Model Checking Output

RQ1. May *small* parameter imprecision lead to *non-negligible* model checking imprecision?

RQ2. Can the asymptotic bounds estimate the imprecise model checking output with satisfactory *accuracy*?

RQ3. Is ProEva's *computational overhead* acceptable?

As the exact bounds are very difficult to compute in general (as explained in Section IV-A), to answer RQ1 and RQ2 we compare the asymptotic bounds with the other two possible forms of bounds, called *linear bounds* and *heuristic bounds*. Let $\vec{v}$ be an optimal solution of Problem (14).

- Linear bounds $\Pi(0) \pm \Pi_1(\vec{v})$ rely on the linear part of the asymptotic expression only.
- Heuristic bounds $\Pi(\vec{v})$ and $\Pi(-\vec{v})$ are computed by heuristically treating $\vec{v}$ as the optimal solution of Problem (5). Notice that the heuristic bounds are *not* in closed form and thus have limited reusability.

Unlike the asymptotic bounds, both the linear bounds and heuristic bounds obviate the computation of the full asymptotic expression. For RQ3, we compare the time of computing asymptotic bounds, linear bounds, heuristic bounds and time-bounded CTMC model checking as the model size increases, and evaluate relationship between the time of computing asymptotic bounds and the number of model parameters. We also show the advantage of asymptotic bounds over heuristic bounds in the function-plotting scenario.

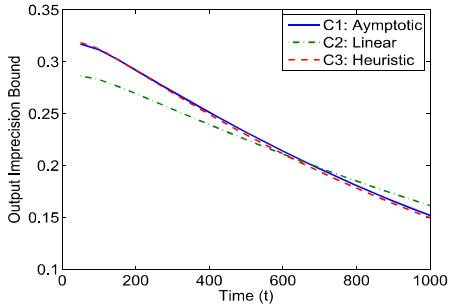
C. Case Study: Fujitsu Disk Drive

Our case study is based on a benchmark CTMC model of a 4-state Fujitsu disk drive [23]. The PRISM specification of the model contains a request queue, a service provider and a power manager. This benchmark model showcases the application of probabilistic model checking to dynamic power management (DPM), which aims to derive a stochastic DPM strategy that minimizes power consumption while satisfying the performance constraints [24]. We analyze how the disk drive model performs (with a given stochastic DPM strategy) against a time-bounded performance metric expressed as "the probability that at least, say, 15 requests are awaiting service by time t." The PRISM specification of the disk drive model manifests eight transition rates, but only four of them can affect the performance of the disk drive (according to the constraint of the power manager). The experiment is performed on a PC with 3.4GHz CPU and 16GB RAM.

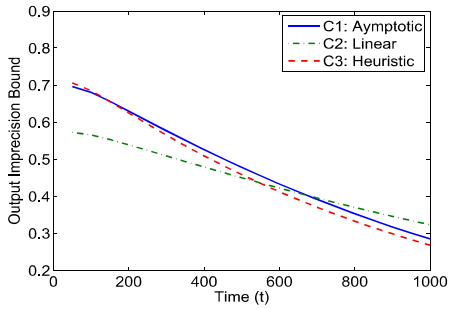
With probabilistic model checking, we can formally validate the cost-effectiveness of stochastic DPM strategies and obviate expensive direct simulation of complex performance metrics. Namely, we only need to measure the four parameters in order to model check those performance metrics. Figure 8 depicts the model checking output that we have reproduced based on the PRISM specification for the aforementioned time-bounded metric, where t ranges from 50 to 1000. However, as explained in Sections I and II, the runtime measurement of the parameters (the transition rates in this case), is likely imprecise. To illustrate the consequence and answer RQ1, we consider two cases that all parameters in the disk drive model are imprecise up to 5% and 10%, respectively. The curves C3 in Figure 9a and Figure 9b present the heuristic bounds for the imprecise output in these two cases. We observe that the output deviates from the result in Figure 8 up to 15-32$\%$ and 29-70$\%$ in the two cases, respectively. Hence, even with small parameter imprecision, the model checking output may mislead any judgement about constraint violation.

To answer RQ2, the curves C1 in Figures 9a and 9b present the asymptotic bounds of the imprecise output. In the case of 5% parameter imprecision, the maximum difference between the asymptotic bounds and the heuristic bounds is less than 1%. In the case of 10% parameter imprecision, the maximum difference between them is less than 2.5%. Thus, when the parameters are imprecise in a small (but non-trivial) scale, we can employ asymptotic bounds as a reasonable estimation of the imprecise model checking output. By contrast, the curves C2 in Figures 9a and 9b representing the linear bounds are much less accurate, with over 3.8% and 10% maximum difference from the heuristic bounds in the two cases, respectively. This justifies the necessity to extend the uniformization method from the linear case to the quadratic case in order to increase accuracy.

To answer RQ3, we present the computation time in Table I for seven CTMC models for the disk drive of different size and a fixed $t = 1000$ (time units). The first three columns of the table present the models, including the model ID, the number of states and the number of transitions. The fourth to eighth columns present the computational time of



(a) 5% Parameter Imprecision



(b) 10% Parameter Imprecision

Figure 9. Accuracy Comparison

Table I
COMPUTATIONAL TIME (SEC.)

model			M_tran	E_comp			Val	PRISM
No.	#stat.	#tran.		CON	LIN	QUA		
1	100	200	0.023	1.213	5.628	20.29		0.160
2	200	400	0.048	1.280	6.196	22.24		0.260
3	400	800	0.096	1.383	7.185	27.14	less	0.490
4	1K	2K	0.167	1.752	10.11	38.45	than	1.182
5	2K	4K	0.500	2.357	15.46	58.41	0.1	2.335
6	5K	10K	1.780	4.497	32.82	130.2		5.728
7	10K	20K	5.403	7.726	59.40	230.3		11.42

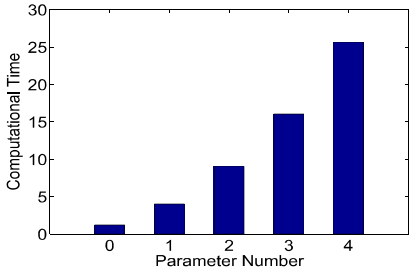


Figure 10. ProEva Computational Time (sec.) vs. Parameter Number

"M_tran", "E_comp" and "Val", respectively. In particular, for "E_comp", the fifth to seventh columns present the time for computing the constant, linear and quadratic parts of the asymptotic expression, respectively. For a comparison, the last column presents the (standard) time-bounded CTMC model checking time using the "sparse" engine of PRISM. The linear bound computation contains "M_tran", "CON" and "LIN", and the quadratic bound computation contains "QUA" in addition. The time of model construction and export with PRISM is not included in the table.

From Table I, we observe that the performance of time-bounded CTMC model checking with ProEva (i.e., "M_tran" and "CON" in "E_comp") is comparable to PRISM. This demonstrates the efficiency of our prototype implementation. Moreover, the ratio of the "QUA" column to the "CON" column and the ratio of the "LIN" column to the "CON" column are roughly stable regardless of the model size increment. This reflects that the computational overhead of asymptotic computation (on time-bounded CTMC model checking) is proportional to the model size.

The efficiency of "Val" for models in all sizes implies its practicality to plot a function relating possible interval-valued parameters and possible imprecise output estimates, which is useful to provide feedback to adjust the parameter measurement in practice. By contrast, since the heuristic bound computation subsumes time-bounded CTMC model checking (with ProEva or PRISM), a high repetition of this method to plot that function may be very inefficient. For example, each computation of a heuristic bound for Model No.7 with PRISM leads to an 11.42-second overhead, which is much higher than the time of "Val". This shows the advantage of computing a closed-form asymptotic expression.

To demonstrate the relationship between the computational time of ProEva and the number of parameters, Figure 10 presents the (average) ProEva computational time where 0 to 4 parameters are selected in Model No.1 in Table I. Notice that if the parameter number is 0, the asymptotic computation is just model checking (i.e., "M_tran" and "CON"). Different from the model size increment, the increment of the parameter number leads to an approximately quadratic increment of the computational time. This is consistent with our analysis of the computation cost in Section IV-B.

In summary, our case study demonstrates the following aspects on the applicability of ProEva, which also provide affirmative answers to RQ1–RQ3.:

- Small-scale imprecise parameters may lead to significant imprecise outputs, which can be accurately approximated by the asymptotic bounds.
- The computational time of ProEva increases roughly quadratically against the number of parameters. When the parameter number is fixed, it is proportional to the standard time-bounded CTMC model checking thereof.
- ProEva is suitable to evaluate (and thus plot) a function relating possible imprecise parameters to possible imprecise model checking outputs.

VI. RELATED WORK AND DISCUSSION

Among the various application domains of probabilistic model checking, ProEva is most relevant to the existing

work applying probabilistic model checking to aid the engineering of self-adaptive software systems [5]. In general terms, a self-adaptive software system is able to modify its behavior and/or structure in response to its perception of the environment and the system itself, and its goals [25]. In general, the adaptation behavior of this system forms a loop with four stages, namely "monitoring", "analyzing", "planning" and "executing" [26]. Calinescu et al. [27] presented a systematic framework QoSMOS based on probabilistic model checking for QoS management and optimization for service-based systems. QoSMOS integrates a suite of tools to support all four stages in the self-adaptation loop. Ghezzi et al. [28] proposed a model-driven framework ADAM consisting of a generator and an interpreter. The generator constructs a probabilistic model of the self-adaptive system from the UML Activity Diagram. The interpreter navigates the invocation of implementation components by searching among alternatives execution paths in the model in order to maximize the system's ability to satisfy its non-functional requirements. Cámara et al. [2] proposed a framework to address the adaptation latencies for self-adaptive systems. Their framework is based on probabilistic model checking of Stochastic Multiplayer Games [29], [30] to derive the best- and worst-case scenarios of adaptation with or without latency, and provides a latency-aware adaptation algorithm to computes the optimal sequence of adaptation decisions.

ProEva is in line with the above-mentioned frameworks for self-adaptive software systems. As a framework of runtime performance evaluation, ProEva fits into the "analyzing" stage of self-adaptation. Similar to the framework of Cámara et al. [2], ProEva is motivated by the configuration latency. Unlike their framework, ProEva formalizes the proactive performance evaluation problem in the presence of configuration latency as a time-bounded CTMC model checking problem. In the three above-mentioned frameworks, only QoSMOS [27] supports CTMC model checking. But to address the challenge of precise parameter measurement, ProEva admits CTMCs with imprecise parameter values, which are beyond the scope of QoSMOS.

One recent technique called parametric (probabilistic) model checking [11], [31] computes a closed-form expression of the model checking output for parametric DTMCs. Filieri et al. [6] presented a parametric model checking framework to support the "analyzing" stage of self-adaptation. Their framework pre-computes such a closed-form expression and evaluate the output efficiently as the probability parameters are (re-)instantiated frequently at runtime. Their framework also infers the partial derivatives of the closed-form expression for sensitivity analysis. Also, Calinescu et al. [32] proposed the framework FACT that employs hill-climbing optimization to propagate the confidence intervals of transition probabilities into a confidence interval of the model checking output.

As explained in Section IV-A, except for highly structured CTMCs, it is impractical to compute closed-form expressions or exact bounds for the output. Therefore, ProEva employs an asymptotic computational technique to achieve a balance between accuracy and computational cost. Our technique is based on, but extends, the existing work [14], [15] on sensitivity analysis of transient distributions for CTMCs. Both ProEva and these works exploit the uniformization method that underpins (standard) time-bounded CTMC model checking [20]. However, while sensitivity analysis is suitable to identify performance-critical parameters by referring to first-order partial derivatives of the transient distribution, in order to achieve reasonably accurate bounds for the model checking output, our technique pursues second-order partial derivatives of the transient distribution. Our technique also extends the previous work [16] that applies the asymptotic analysis to DTMC model checking. Han et al. [33] considered the parametric synthesis for CTMCs, which is orthogonal to our problem. Their method discretizes a continuous region and obtains finite sample points, and thus is also quite different from our asymptotic approach. Brázidil et al. [34] considered a different parameter synthesis problem for a "fixed-delay" variant of CTMCs. Hajiaghayi et al. [35] analyzed possibly infinite CTMCs using simulation, while ProEva deals with finite CTMCs only and employs numerical computation.

VII. Conclusions

We have presented ProEva, a framework of runtime proactive performance evaluation. The core method of ProEva handles time-bounded CTMC model checking in the presence of imprecise transition rates. Our empirical evaluation demonstrates that, with an acceptable computational overhead on standard time-bounded CTMC model checking, ProEva can accurately characterize the output imprecision.

We plan to extend the framework of ProEva to evaluate a broader range of performance metrics expressed by the full-fledged syntax of CSL in future. Another important research direction is to incorporate a suitable parameter estimation method into ProEva.

Acknowledgment

This work was partially supported by Singapore Ministry of Education (MOE2015-T2-1-137), National Natural Science Foundation of China (61502260), UK EPSRC grant (EP/P00430X/1), Australian Research Council (DP160101652 and DP130102764), and the CAS/SAFEA International Partnership Program for Creative Research Team. T. Chen is also supported by European CHIST-ERA project SUCCESS, NSFC grant (61662035), and an overseas grant from the State Key Laboratory of Novel Software Technology at Nanjing University (KFKT2014A14).

REFERENCES

[1] Amazon. (2016) Amazon Cloudwatch documentation. [Online]. Available: http://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/WhatIsCloudWatch.html

[2] J. Cámara, G. A. Moreno, D. Garlan, and B. Schmerl, "Analyzing latency-aware self-adaptation using stochastic games and simulations," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 10, no. 4, p. 23, 2016.

[3] G. Bolch, S. Greiner, H. de Meer, and K. S. Trivedi, *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications*. New York, NY, USA: Wiley-Interscience, 1998.

[4] J.-P. Katoen, "The probabilistic model checking landscape," in *Proceedings of the 31th Annual ACM/IEEE Symposium on Logic in Computer Science*, ser. LICS'16, 2016.

[5] R. Calinescu, C. Ghezzi, M. Kwiatkowska, and R. Mirandola, "Self-adaptive software needs quantitative verification at runtime," *Commun. ACM*, vol. 55, no. 9, pp. 69–77, Sep. 2012.

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

[7] G. Su, T. Chen, Y. Feng, D. S. Rosenblum, and P. S. Thiagarajan, "An iterative decision-making scheme for Markov Decision Processes and its application to self-adaptive systems," in *Proceedings of the 19th International Conference on Fundamental Approaches to Software Engineering*, ser. FASE'16. Eindhoven, the Netherlands: Springer, 2016.

[8] G. Su, D. S. Rosenblum, and G. Tamburrelli, "Reliability of run-time Quality-of-Service evaluation using parametirc model checking," in *Proceedings of the 38th International Conference on Software Engineering*, ser. ICSE'16. New York, NY, USA: ACM, 2016.

[9] T. Zheng, C. M. Woodside, and M. Litoiu, "Performance model estimation and tracking using optimal filters," *IEEE Transactions on Software Engineering*, vol. 34, no. 3, pp. 391–406, 2008.

[10] T. Chen, T. Han, and M. Z. Kwiatkowska, "On the complexity of model checking interval-valued discrete time Markov chains," *Inf. Process. Lett.*, vol. 113, no. 7, pp. 210–216, 2013.

[11] C. Daws, "Symbolic and parametric model checking of discrete-time Markov chains," in *Proceedings of the First International Conference on Theoretical Aspects of Computing*, ser. ICTAC'04. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 280–294.

[12] K. Sen, M. Viswanathan, and G. Agha, "Model-checking Markov chains in the presence of uncertainties," in *Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2006, pp. 394–410.

[13] T. Chen, M. Diciolla, M. Z. Kwiatkowska, and A. Mereacre, "Time-bounded verification of CTMCs against real-time specifications," in *Formal Modeling and Analysis of Timed Systems - 9th International Conference, FORMATS 2011, Aalborg, Denmark, September 21-23, 2011. Proceedings*, ser. Lecture Notes in Computer Science, U. Fahrenberg and S. Tripakis, Eds., vol. 6919. Springer, 2011, pp. 26–42.

[14] P. Heidelberger and A. Goyal, "Sensitivity analysis of continuous time markov chains using uniformization," in *Computer Performance and Reliability*. Elsevier Science Publishers B.V., 1988, pp. 93–104.

[15] H. Abdallah, "Sensitivity computation of reliability markov models using the uniformized power method," *Reliability Engineering & System Safety*, vol. 56, no. 1, pp. 53–59, 1997.

[16] G. Su, Y. Feng, T. Chen, and D. S. Rosenblum, "Asymptotic perturbation bounds for probabilistic model checking with empirically determined probability parameters," *IEEE Trans. Software Eng.*, vol. 42, no. 7, pp. 623–639, 2016.

[17] M. Kwiatkowska, G. Norman, and D. Parker, "PRISM 4.0: Verification of probabilistic real-time systems," in *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*. Springer, 2011, pp. 585–591.

[18] C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen, "Model-checking algorithms for continuous-time Markov chains," *Software Engineering, IEEE Transactions on*, vol. 29, no. 6, pp. 524–541, 2003.

[19] V. G. Kulkarni, *Modeling and Analysis of Stochastic Systems*. London, UK, UK: Chapman & Hall, Ltd., 1995.

[20] W. J. Stewart, *Introduction to the numerical solutions of Markov chains*. Princeton Univ. Press, 1994.

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

[22] G. Su, T. Chen, Y. Feng, and D. S. Rosenblum, "ProEva: Runtime proactive performance evaluation based on continuous-time Markov chains." available on http://comp.nus.edu.sg/~sugx/ProEva/full-version.pdf, 2016.

[23] Q. Qiu, Q. Wu, and M. Pedram, "Stochastic modeling of a power-managed system: construction and optimization," in *Proc. International Symposium on Low Power Electronics and Design*, 1999.

[24] G. Norman, D. Parker, M. Kwiatkowska, S. K. Shukla, and R. K. Gupta, "Formal analysis and validation of continuous-time markov chain based system level power management strategies," in *High-Level Design Validation and Test Workshop, 2002. Seventh IEEE International*. IEEE, 2002, pp. 45–50.

[25] R. de Lemos, H. Giese, H. A. Müller, and *et al.*, "Software engineering for self-adaptive systems: A second research roadmap," in *Software Engineering for Self-Adaptive Systems II*, R. de Lemos, H. Giese, H. A. Müller, and M. Shaw, Eds. Springer-Verlag, 2013, vol. 7475, pp. 1–32.

[26] M. C. Huebscher and J. A. McCann, "A survey of autonomic computing—degrees, models, and applications," *ACM Comput. Surv.*, vol. 40, no. 3, pp. 7:1–7:28, Aug. 2008.

[27] R. Calinescu, L. Grunske, M. Z. Kwiatkowska, R. Mirandola, and G. Tamburrelli, "Dynamic qos management and optimization in service-based systems," *IEEE Trans. Software Eng.*, vol. 37, no. 3, pp. 387–409, 2011.

[28] C. Ghezzi, L. S. Pinto, P. Spoletini, and G. Tamburrelli, "Managing non-functional uncertainty via model-driven adaptivity," in *Proceedings of the 2013 International Conference on Software Engineering*, ser. ICSE'13. IEEE Press, 2013, pp. 33–42.

[29] T. Chen, V. Forejt, M. Z. Kwiatkowska, D. Parker, and A. Simaitis, "Automatic verification of competitive stochastic systems," *Formal Methods in System Design*, vol. 43, no. 1, pp. 61–92, 2013.

[30] ——, "Prism-games: A model checker for stochastic multi-player games," in *Tools and Algorithms for the Construction and Analysis of Systems - 19th International Conference, TACAS 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings*, ser. Lecture Notes in Computer Science, N. Piterman and S. A. Smolka, Eds., vol. 7795. Springer, 2013, pp. 185–191.

[31] 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.

[32] R. Calinescu, C. Ghezzi, K. Johnson, M. Pezzé, Y. Rafiq, and G. Tamburrelli, "Formal verification with confidence intervals: A new approach to establishing the Quality-of-Service properties of software systems," *Reliability, IEEE Transactions on*, 2015.

[33] T. Han, J.-P. Katoen, and A. Mereacre, "Approximate parameter synthesis for probabilistic time-bounded reachability," in *Real-Time Systems Symposium*, 2008, pp. 173–182.

[34] T. Brázdil, L. Korenčiak, J. Krčál, P. Novotný, and V. Řehák, "Optimizing performance of continuous-time stochastic systems using timeout synthesis," in *12th International Conference on Quantitative Evaluation of Systems*, 2015.

[35] M. Hajiaghayi, B. Kirkpatrick, L. Wang, and A. Bouchardcôté, "Efficient continuous-time Markov chain estimation," in *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, 2014, pp. 638–646.

[36] T. Chen, Y. Feng, D. S. Rosenblum, and G. Su, "Perturbation analysis in verification of discrete-time Markov chains," in *Proceedings of the 25th International Conference on Concurrency Theory (CONCUR'14)*, 2014.

[37] T. Chen, T. Han, J. Katoen, and A. Mereacre, "Model checking of continuous-time Markov chains against timed automata specifications," *Logical Methods in Computer Science*, vol. 7, no. 1, 2011.