University of Wollongong

Research Online

Faculty of Engineering and Information
Sciences - Papers: Part B

Faculty of Engineering and Information
Sciences

2020

Quantitative Verification for Monitoring Event-Streaming Systems

Guoxin Su

University of Wollongong, guoxin@uow.edu.au

Li Liu

Minjie Zhang

University of Wollongong, minjie@uow.edu.au

David Rosenblum

david@comp.nus.edu.sg

Follow this and additional works at: <https://ro.uow.edu.au/eispapers1>

Part of the [Engineering Commons](#), and the [Science and Technology Studies Commons](#)

Research Online is the open access institutional repository for the University of Wollongong. For further information contact the UOW Library: research-pubs@uow.edu.au

Quantitative Verification for Monitoring Event-Streaming Systems

Abstract

IEEE High-performance data streaming technologies are increasingly adopted in IT companies to support the integration of heterogeneous and possibly distributed service systems and applications. Compared with the traditional message queuing middleware, a streaming platform enables the implementation of event-streaming systems (ESS) which include not only complex queues but also applications that transform and react to the streams of data. By analysing the centralised data streams, one can evaluate the Quality-of-Service for other systems and components that produce or consume the streams. We consider the exploitation of probabilistic model checking as a performance monitoring technique for ESS systems. Probabilistic model checking is a mature, powerful verification technique with successful application in performance analysis. However, an ESS system may contain quantitative parameters that are determined by event streams observed in a certain period of time. In this paper, we present a novel theoretical framework called QV4M (meaning “quantitative verification for monitoring”) for ESS system monitoring based on two recent methods of probabilistic model checking. Our framework QV4M assumes the parameters in a probabilistic system model as random variables and infers the statistical confidence of a probabilistic model checking output. We present two case studies as an empirical evaluation of QV4M.

Disciplines

Engineering | Science and Technology Studies

Publication Details

Su, G., Liu, L., Zhang, M. & Rosenblum, D. (2020). Quantitative Verification for Monitoring Event-Streaming Systems. IEEE Transactions on Software Engineering,

Quantitative Verification for Monitoring Event-Streaming Systems

Guoxin Su, *Member, IEEE*, Li Liu, Minjie Zhang, *Senior Member, IEEE* and David S. Rosenblum, *Fellow, IEEE*

Abstract—High-performance data streaming technologies are increasingly adopted in IT companies to support the integration of heterogeneous and possibly distributed applications. Compared with the traditional message queuing middleware, a streaming platform enables the implementation of event-streaming systems (ESS) which include not only complex queues but also pipelines that transform and react to the streams of data. By analysing the centralised data streams, one can evaluate the Quality-of-Service for other systems and components that produce or consume those streams. We consider the exploitation of *probabilistic model checking* as a performance monitoring technique for ESS systems. Probabilistic model checking is a mature, powerful verification technique with successful application in performance analysis. However, an ESS system may contain quantitative parameters that are determined by event streams observed in a certain period of time. In this paper, we present a novel theoretical framework called QV4M (meaning “quantitative verification for monitoring”) for monitoring ESS systems, which is based on two recent methods of probabilistic model checking. QV4M assumes the parameters in a probabilistic system model as random variables and infers the statistical significance for the probabilistic model checking output. We also present an empirical evaluation of computational time and data cost for QV4M.

Index Terms—Discrete-time Markov chain, event stream, parametric model checking, performance monitoring, probabilistic model checking, statistical inference

1 INTRODUCTION

RECENT years witness the popularity of streaming platforms (e.g., Apache Kafka, MapR Streaming and Azure Service Bus) as data flow backbones in data-intensive distributed systems. Compared with the traditional message queuing middleware, streaming platforms provide a set of high-level APIs to implement pipelines that produce, transform, and consume or process event streams. Figure 1 illustrates the architecture of an *event-streaming system* (ESS), which includes a software system (i.e., the event source) and a streaming pipeline. There are various use cases with event streams, including log searching, event store and advanced analytics. In particular, we can replay system executions from event streams and attain in-depth understanding on system performance.

Roughly speaking, *performance metrics*, or Quality-of-Service (QoS) metrics, can be atomic or composite. Atomic performance metrics, such as “link failure”, “delay” and “utilisation”, can be measured directly. Composite performance metrics, such as “average availability”, “maximum response time” and “top 10%”, are the aggregations of atomic performance metrics. One important kind of com-

Fig. 1. Event-streaming system (ESS)

posite performance metrics are the *behavioural* (or *temporal*) metrics, whose examples include “failure rate in ten consecutive requests” and “expected time to reach maximum queue size”.

We aim to leverage *probabilistic model checking* [21], which is a well established verification technique for stochastic systems, to monitor the complex behavioural metrics for ESS systems. As illustrated in Figure 2, probabilistic model checking accepts a stochastic system model (such as a discrete-time Markov chain (DTMC)) and one or more properties in a temporal logic (such as Probabilistic Reward Computation Tree Logic (PRCTL)) as input, and produces the verification output expressing “whether or to what extent those properties are satisfied by the system”. A number of probabilistic model checking tools (e.g., PRISM [23], Storm [11], PARAM [17] and PAT [31], among others) have been developed. Probabilistic model checking has been successfully applied to a variety of domains, including communication protocol analysis, probabilistic programs and systems biology. Recently, this technique is also utilised in the runtime QoS evaluation of self-adaptive software systems where the system parameters may be updated

- G. Su is with the School of Computing and Information Technology, University of Wollongong, Australia.
E-mail: guoxin@uow.edu.au
- L. Liu is with School of Big Data & Software Engineering, Chongqing University, Chongqing 400044, PR China
E-mail: dcsliliu@cqu.edu.cn
- M. Zhang is with the School of Computing and Information Technology, University of Wollongong, Australia.
E-mail: minjie@uow.edu.au
- D. S. Rosenblum is with Department of Computer Science, School of Computing, National University of Singapore, Singapore.
E-mail: david@comp.nus.edu.sg

Manuscript received xxx; revised xxx; accepted xxx.

Fig. 2. Probabilistic model checking

continuously [4], [7], [16]. A broad range of behavioural metrics related to the system adaptation objectives can be formalised and analysed by using this technique.

In an ESS system, performance metrics usually involve parameters that must be estimated by observing an event stream. For example, to monitor “the failure rate in any ten consecutive requests”, we need to count the number of failures in all the requests collected in a specific time frame. Thus, in order to leverage probabilistic model checking for ESS performance monitoring, it is crucial to account for the *statistical significance* problem, namely:

Whether the quantitative verification output is statistically significant?

Various recent methods of probabilistic model checking have been developed for stochastic systems in the presence of imprecise probabilities. Among these methods, parameter synthesis [17], [25] is an active recent research topic. This approach relies on a symbolic computation technique called parametric model checking, [9] which computes a closed-form rational function for all possible output values. Different from the symbolic computation, another approach called asymptotic perturbation analysis [6], [29] extends the standard matrix-iteration method of probabilistic model checking to compute the partial derivatives of an output value against the perturbed parameters. However, these approaches do not address the aforementioned statistical significance problem. Calinescu et al. [3] presented a framework FACT, which can propagate the confidence intervals from the unknown transition probabilities of a parametric DTMC to the verification output. But FACT exploits parametric model checking and global optimisation of rational functions, which is highly computationally expensive.

We present a novel performance monitoring framework called QV4M (which stands for “quantitative verification for monitoring”) for ESS systems. In QV4M, the model parameters are not fixed but need to be estimated by statistics of event streams. Besides computing the output of quantitative verification, QV4M addresses the statistical significance for the output against a performance baseline. To achieve this, QV4M employs two existing methods in probabilistic model checking (i.e., matrix-iteration and parametric model checking) to compute the partial derivatives of the output (with respect to model parameters) and infers p -values for the output. Similar to other quantitative verification frameworks, we consider the computation time of QV4M with respect to the model size in our empirical evaluation. In addition to the time cost, we also consider the data cost of QV4M, namely, the relationship between the sample size and the p -values. This work extends the monitoring framework published in

Fig. 3. Microservice orchestration system

our conference paper [30].

The remainder of the paper is organised as follows. Section 2 describes a running example and presents the background and motivation for our framework. Section 3 presents the basics of probabilistic model checking. Section 4 presents the formal monitoring problem and a preview of our framework. Section 5 presents the techniques in detail, including two computational methods and a statistical inference method. Section 6 presents a statistical inference method of multi-topic streams. Section 7 presents the case studies. Section 8 discusses the related work. Finally, Section 9 concludes the paper.

2 EXAMPLE AND MOTIVATION

2.1 Example: Microservice Orchestration System

We present a microservice orchestration system, which is inspired by the Netflix Conductor¹ (an open-source workflow engineer to orchestrate the Netflix microservices), as a motivating example. The architecture of this system comprises a Workflow Server, a pool of Workers and a Shared File System, as depicted in Figure 3. The Workflow Server orchestrates the workflow tasks, puts them in the queues and keeps track of their progress. There are three internal components in the Workflow Server: The Workflow & Task Service starts and manages the workflows; the Storage service stores the workflow blueprints and individual task specifications; and the Queue Service schedules the tasks in an appropriate queue. In a usual production environment, many instances of the same or different workflows run concurrently. The Workers (which may run in virtual machines (VM) or containers) poll the tasks from the queues, process them and update the status to the workflow server. When a Worker processes a task, it reads the input from and writes the output to the Shared File System.

Communications between services in this microservice orchestration system create different (types of) operation events (as indicated in Figure 3), including workflow lifecycle events (e.g., “start workflow”, “get workflow def”, “schedule task”, “put in queue”, “poll for task”, and “update task status”), queuing events (e.g., “put in queue” and “poll for task”) and IO events (e.g., “reading” and “writing”). The microservice system is integrated with a streaming pipeline implemented a platform such as Apache

1. <https://netflix.github.io/conductor/>

Kafka [12]. All events of the microservice orchestration system are ingested a common Kafka stream by a generic event producer in Java (Figure 4), which appends the events (i.e., records) in the form of key-value pairs to the stream (named “all_events”). In the streaming pipeline, this input stream can be transformed into different other streams, which can be consumed and processed in different use cases (such as log searching, external storage and advanced analytics). The microservice orchestration system and the streaming pipeline constitute a realistic example ESS system.

2.2 Quantitative Verification of QoS Metrics

Among the aforementioned use cases, we consider performance analysis. To facilitate Worker management, suppose the Workers in our microservice orchestration system are grouped into a running pool and a sleeping pool. More Workers in the running pools can lead to better service quality but with higher operation cost. To be cost-effective, the system needs to access to a variety of performance metrics, such as “the average queuing time”, “the maximum IO rate”, “the average workflow execution time” and “the successful workflow completion rate if the maximum retry number of tasks is 3”. All of these metrics can be measured by processing the event streams of the system.

With probabilistic model checking, we can build different models for our ESS system and analyse various complex behavioural metrics. To demonstrate this, we elaborate a model built by using the aforementioned queuing events of tasks. (Our empirical evaluation in Section 7 also considers a model for the workflow lifecycle events.) To make the problem more interesting, we further assume that all tasks are categorised as low or high priority, and therefore the task queue is a *priority queue* essentially. The Workers can poll for low-priority tasks only if no high-priority tasks is awaiting service. A queuing event stream can be transformed from the input stream by filtering queuing events in the Kafka pipeline only. At an endpoint of the pipeline, the queuing event stream is consumed and aggregated, and some event counters are generated for our purpose (Figure 5).

To apply probabilistic model checking, we first need to build a formal model for the task queue. Figure 6 presents a model specification in the language of the probabilistic model checking tool PRISM [23]. This PRISM model is essentially a parametric DTMC. The main body of this model includes two state variables (HP and LP), which represent the high-priority and low-priority tasks in the queue, respectively, and a set of state transition rules, which involve four quantitative parameters, namely two enqueue ratios (p and q), a dequeue ratio ($r = 1 - p - q$) and the maximum of awaiting tasks (N) for both high-priority and low-priority tasks. The PRISM model also includes a reward (or cost) structure, which assigns an integer cost (t) to all states. The value of N is defined by the user. Based on the returned values of event counters in Figure 5, the ratios p , q and r and the cost t are calculated. In particular, assuming $in1, in2, out > 0$, let $p = in1/(in1 + in2 + out)$, $q = in2/(in1 + in2 + out)$ and t can be a scaled and rounded value of $(in1 + in2 + out)/60s$.

Once the model is built, we can formalise a very broad range of properties, which are interesting behavioural metrics to analyse, as temporal logic formulas (whose formal

```
1 public class GenericEventProducer {
2   public static Properties props = new Properties();
3   public static KafkaProducer<String, String> producer = new
      KafkaProducer<String, String>(props);
4   public void produce(String key, String value) {
5     ProducerRecord<String, String> record = new
      ProducerRecord<>("all_events", key, value);
6     producer.send(record);
7   }
8 }
```

Fig. 4. Generic event producer in the Kafka streaming pipeline

```
1 public class QueuingEventAggregator {
2   public static Properties props = new Properties();
3   public static KafkaConsumer<String, String> consumer =
      new KafkaConsumer<String, String>(props);
4   public static void countEvents() {
5     consumer.subscribe("queuing_events")
6     while (true) {
7       ConsumerRecords<String, String> records =
          consumer.poll(60); //poll for records every 60 seconds
8       long out, in1, in2; // counters
9       for (ConsumerRecord<String, String> rec: records) {
10        if (rec.key() == "poll") {out += 1;}
11        else if (rec.value() == "high") {in1 += 1;}
12        else if (rec.value() == "low") {in2 += 1;}
13        // more lines ...
14      }
15    }
16 }
```

Fig. 5. Queuing event aggregator in the Kafka streaming pipeline

syntax is presented in Section 4). Two examples of behavioural metrics are formalised as follows:

- $\Phi_1 \triangleq P_{=?}[(HP < K) \vee (LP = K)]$
- $\Phi_2 \triangleq C_{=?}[\mathbb{F}((HP = K) \vee (LP = K))]$

where $K \leq N$. Intuitively, the above two formulas express the following two queries:

- How likely will the number of low-priority tasks reach K whilst the number of high-priority tasks is always less than K ?
- What is the average time (units) until the number of tasks of either kind reaches K ?

The results (say, res_1 and res_2) of the above two queries can be computed continuously (i.e., every 60 seconds).

2.3 Challenges for Monitoring

In monitoring, we need to compare the computed metrics against the prescribed values in a performance baseline. We have explained that probabilistic model checking is powerful in expressing and analysing a broad range of behavioural metrics. However, as the quantitative parameters in an ESS model (such as p, q, t in the priority queue) are determined (or estimated) by events collected in a time frame, we need to consider whether the difference between a computed result of a metric (say, Φ_1 or Φ_2) and its baseline value is statistically significant. The existing quantitative verification frameworks provide little account for this problem. Moreover, quantitative verification is computation-intensive technique in general but latency is a crucial factor in monitoring. In the priority queue model in Figure 6, the only non-fixed part of the model is the set of quantitative parameters. Thus, it is desirable to reuse some intermediate computation results in order to improve the overall efficiency of repeated quantitative verification.


```

dtmc // priority queue//
const double p; //incoming ratio for h.p. tasks//
const double q; //incoming ratio for l.p. tasks//
const double r=1-p-q; //polling ratio//
const int N; // max number of tasks//
const int t; // avg time (as an integer) between events//
module Queues
  HP: [0..N]; // number of h.p. tasks//
  LP: [0..N]; // number of l.p. tasks//
  [] HP=0 & LP=0 -> p/(p+q):(HP'=HP+1)
    + q/(p+q):(LP'=LP+1);
  [] HP=0 & LP>0 & LP<N -> p:(HP'=HP+1)
    + q:(LP'=LP+1) + r:(LP'=LP-1);
  [] HP=0 & LP=N -> p/(p+r):(HP'=HP+1)
    + r/(p+r):(LP'=LP-1);
  [] HP>0 & HP<N & LP<N -> p:(HP'=HP+1)
    + q:(LP'=LP+1) + r:(HP'=HP-1);
  [] HP>0 & HP<N & LP=N -> p/(p+r):(HP'=HP+1)
    + r/(p+r):(HP'=HP-1);
  [] HP=N & LP<N -> q/(q+r):(LP'=LP+1)
    + r/(q+r):(HP'=HP-1);
  [] HP=N & LP=N -> 1:(HP'=HP-1);
endmodule
rewards
  [] true : t;
endrewards

```

Fig. 6. ESS model in PRISM: Priority queue

In view of this, we propose a new framework of probabilistic model checking, which is tailored to performance monitoring for ESS system. This new framework should be able to (i) integrate with the event stream statistics, (ii) infer the statistical significance of the output, and (iii) separate the pre-computation phase and the online phase of quantitative verification [15].

3 BASICS OF PROBABILISTIC MODEL CHECKING

In this section, we recall the basics of probabilistic model checking.

3.1 Parametric Discrete-Time Markov Chain

Definition 1 (DTMC). A DTMC with rewards is the tuple:

$$\mathcal{M} = (S, s_0, \mathbf{P}, L, R_1, R_2)$$

where

- S is a finite, non-empty state space with $s_0 \in S$ being an initial state,
- $\mathbf{P} : S \times S \rightarrow [0, 1]$ is a transition probability function (also called a transition matrix) such that, for each $s \in S$, $\mathbf{P}(s, \cdot)$ is an exit distribution, namely, $\sum_{t \in S} \mathbf{P}(s, t) = 1$,
- $L : S \rightarrow 2^{AP}$ assigns a subset of atomic propositions $L(s) \subseteq AP$ to each state s ,
- $R_1 : S \rightarrow \mathbb{N}$ assigns a reward (or cost) $R_1(s)$ to each state s , and
- $R_2 : S \times S \rightarrow \mathbb{N}$ assigns a reward (or cost) $R_2(s)$ to each transition (s, t) .

For two transition matrices $\mathbf{P}$ and $\mathbf{P}'$ of the same dimension, we write $\mathbf{P} \sim \mathbf{P}'$ if and only if they have exactly the same positions of zero (or non-zero) entries.

A key component of a parametric DTMC is a *parameteric transition matrix* in a DTMC. We introduce a vector of variables $\mathbf{x} = (x_1, \dots, x_m)$ for some $m > 0$. Let $\mathbf{P}_{\mathbf{x}}$ be

Fig. 7. Parametric DTMC of a simple priority queue

$$\begin{bmatrix}
 0 & \frac{p}{p+q} & 0 & 0 & \frac{q}{p+q} & 0 & 0 & 0 & 0 \\
 r & 0 & p & 0 & q & 0 & 0 & 0 & 0 \\
 0 & \frac{r}{q+r} & 0 & 0 & 0 & \frac{q}{q+r} & 0 & 0 & 0 \\
 r & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & 0 & 0 & p & q & 0 & 0 \\
 0 & 0 & 0 & r & 0 & 0 & p & q & 0 \\
 0 & 0 & 0 & 0 & \frac{r}{q+r} & 0 & 0 & 0 & \frac{q}{q+r} \\
 0 & 0 & 0 & \frac{r}{p+r} & 0 & 0 & 0 & \frac{p}{q+r} & 0 \\
 0 & 0 & 0 & 0 & 0 & 0 & \frac{r}{p+r} & 0 & \frac{p}{p+r} \\
 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0
 \end{bmatrix}$$

Fig. 8. Parametric transition matrix for the simple priority queue

a *parametric transition matrix* of $\mathbf{P}$ resulted from replacing the constant probabilities in some entries of $\mathbf{P}$ with *rational functions* of $\mathbf{x}$. Recall that a rational function is a fraction of two polynomial functions. We require the parameter vector $\mathbf{x}$ is contained in an *open set* $\mathcal{D}$, which is a subset of $(0, 1)^m$ such that (i) $\mathbf{P}_{\mathbf{x}}$ is a transition matrix (as per Def. 1) and (ii) $\mathbf{P}_{\mathbf{x}} \sim \mathbf{P}$ for all $\mathbf{x} \in \mathcal{D}$. An example of the parametric transition matrix for our task queues model is presented in Figure 8, where the domain of parameters is $\{(p, q) \in (0, 1)^2 \mid p + q < 1\}$.

Definition 2 (Parametric DTMC). A parametric DTMC with rewards is the following tuple:

$$\mathcal{M}_{\mathbf{x}} = (S, s_0, \mathbf{P}_{\mathbf{x}}, L, R_1, R_2)$$

where $\mathbf{P}_{\mathbf{x}}$ is a parametric transition matrix of $\mathbf{P}$ and the other components are defined as in Definition 1.

Figure 7 depicts a parametric DTMC for the task queue whose the maximum queue is 2 (i.e., $N = 2$). The model has 9 states, each of which is associated with a label $(i_1 i_2)$, where i and j are the numbers of jobs in the low- and high-priority queues, respectively. The model has two 3 parameters p and q , which also appear in the parametric transition matrix in Figure 8. Note that $r = 1 - p - q$, and thus r is not an (independent) parameter.

A *path* in $\mathcal{M}_{\mathbf{x}}$ is an infinite sequence of states $\pi = s_0 s_1 \dots$ such that $\mathbf{P}_{\mathbf{x}}(s_i, s_{i+1}) > 0$ for all $i \in \mathbb{N}$ and all $\mathbf{x} \in \mathcal{D}$. Let $\pi[i]$ denote the $(i + 1)^{\text{th}}$ state in π . For each $s \in S$, let $\text{Path}^{\mathcal{M}_{\mathbf{x}}}(s)$ denote the set of paths π in $\mathcal{M}_{\mathbf{x}}$ such that $\pi[0] = s$. A *probability measure* $Pr^{\mathcal{M}_{\mathbf{x}}}$ for $\mathcal{M}_{\mathbf{x}}$ on $\text{Path}^{\mathcal{M}_{\mathbf{x}}}(s)$ can be defined (c.f., Baier and Katoen [1, Ch. 10]). For simplicity, we just write the path space and the probability measure of $\mathcal{M}_{\mathbf{x}}$ as Path and Pr , respectively.

3.2 Temporal Logic and Semantics

We define a fragment of Probabilistic Reward Computation Tree Logic (PRCTL) as follows:

$$\begin{aligned}\Phi &::= P_{=?}[\phi] \mid C_{=?}[FA] \\ \phi &::= XA \mid AU A \mid AU^n A \\ A &::= aa \mid \neg A \mid A \wedge A \mid tt\end{aligned}$$

where $aa \in AP$, tt the tautology, $n \in \mathbb{N}$, X is the “next” operator, U is the “until” connective and $U^{\leq n}$ is the “until-within- n -steps” connective. We call $P_{=?}[\phi]$ a *probability query formula*, which informally expresses the query of the probability that ϕ is satisfied. We call $C_{=?}[A]$ a *reward query formula*, which informally expresses the query of the accumulated reward for reaching states at which A is satisfied. Following the literature, let $FA = tt \vee A$ and $F^{\leq n} A = tt \vee U^{\leq n} A$.

We now present the semantics of our temporal logic. The semantic relation $\mathcal{M}_x, s \models A$ (or just $s \models A$) is standard as in proposition logic. The semantic relation $\mathcal{M}_x, \pi \models \phi$, which intuitively means “ ϕ is satisfied in the path π ”, is defined recursively as follows:

- $\mathcal{M}_x, \pi \models XA$ iff $\pi[1] \models A$.
- $\mathcal{M}_x, \pi \models AU A'$ iff $\exists n > 0$ s.t. $\pi[n'] \models A'$ & $\pi[j] \models A, \forall 0 \leq j < n$.
- $\mathcal{M}_x, \pi \models AU^{\leq n} A'$ iff $\exists 0 < n' \leq n$ s.t. $\pi[n'] \models A'$ & $\pi[j] \models A, \forall 0 \leq j < n'$.

Next, we define that

$$R_A(\pi) = \sum_{i=0}^n R_1(s_i) + \sum_{i=0}^{n-1} R_2(s_i, s_{i+1})$$

where n is the least number such that $s_n \models A$, if it exists. If such n does not exist, then let $R_A(\pi) = \infty$. Given $\mathcal{M}_x$ and $s \in S$, the semantics of $P_{=?}[\phi]$ and $C_{=?}[A]$ are functions defined as follows.

- $\llbracket P_{=?}[\phi] \rrbracket_s^{\mathcal{M}_x} : \mathcal{D} \rightarrow [0, 1]$ such that $\forall x \in \mathcal{D}$:
 $\llbracket P_{=?}[\phi] \rrbracket_s^{\mathcal{M}_x}(x) = \Pr\{\pi \in \text{Path}(s) \mid \mathcal{M}_x, \pi \models \phi\};$
- $\llbracket C_{=?}[A] \rrbracket_s^{\mathcal{M}_x} : \mathcal{D} \rightarrow \mathbb{R}_{\geq 0} \cup \{\infty\}$ such that $\forall x \in \mathcal{D}$:
 $\llbracket C_{=?}[A] \rrbracket_s^{\mathcal{M}_x}(x) = \begin{cases} \sum_{i=0}^{\infty} i \cdot \Pr\{\pi \in \text{Path}(s) \mid i = R_A(\pi)\} \\ \text{if } R_A(\pi) < \infty, \forall \pi \in \text{Path}(s) \\ \infty \text{ otherwise.} \end{cases}$

Definition 3 (Verification output function). *Given $\mathcal{M}_x$ and Φ , a verification output function (VOF) is defined as follows:*

$$f : x \mapsto \llbracket \Phi \rrbracket_{s_0}^{\mathcal{M}_x}(x) \in \mathbb{R}, \quad x \in \mathcal{D}.$$

Based on the definition of $\mathcal{D}$ (c.f., Section 3.1), f is smooth (and thus differentiable) in $\mathcal{D}$. Let $h \in \mathbb{R}$, which represents a performance baseline or threshold. The *formal problem of quantitative verification* is to determine whether $f(x) < h$ or $f(x) > h$ for a given $x \in \mathcal{D}$. Probabilistic model checking provides practical methods (such as the matrix-iteration method and parametric model checking) computes a VOF value $f(x)$.

4 FORMAL PROBLEM AND FRAMEWORK

In this section, we formalise the monitoring problem and present an overview of the QV4M framework.

4.1 Stream Modeling and Random Sampling

Let $\mathcal{E} = \{e_1, \dots, e_m\}$ be a finite non-empty set of *events* such that $m = |\mathcal{E}|$. (The number of event types equals the number of parameters in a parametric DTMC.) An *event stream* (or simply a *stream*) λ is a finite sequence of events, namely $\lambda \in \mathcal{E}^*$. We introduce a *Bernoulli random variable* X_i for each $1 \leq i \leq |\mathcal{E}|$, which is defined as follows:

$$X_i : \mathcal{E} \rightarrow \{0, 1\}$$

such that $X_i(e_j) = 0$ if $j \neq i$ and $X_i(e_i) = 1$ otherwise. Also let $\mathbf{X} = (X_i)_{1 \leq i \leq m}$. The following assumption associates Bernoulli random variables with the parametric DMTC parameters. It also allows us to infer the statistical significance of our quantitative verification output when the parameter values are determined.

Assumption 1. *A stream λ is a random sample of $\mathbf{X}$.*

The sampling data and events for the priority queue model are as follows:

Sample data	Event type
(1, 0)	A: put high-priority task in queue
(0, 1)	B: put low-priority task in queue
(0, 0)	C: poll for task from queue

The two parameters p and q in the task queues are estimated by the frequencies of the three queuing event types. Let X_p and X_q be the Bernoulli random variables that are associated with p and q , respectively. The outcome of X_p (resp., X_q) is 1 *if and only if* an instance of Event A (resp., Event B) occurs in the queuing event stream. After observing three event types in a period, we use the mean of X_p (resp., X_q) to estimate p (resp., q) whereas the expected value of X_p (resp., X_q) is just the true value of p (resp., q). We can also see a strong correlation of p and q : At any point of time, *only* one of the three event types can happen.

4.2 The Monitoring Problem

Recall that, given a VOF f , a threshold h and a vector $\mathbf{x}$ of parameter values, quantitative verification determines whether $f(\mathbf{x}) > h$ or $f(\mathbf{x}) < h$. As “ $>$ ” and “ $<$ ” are dual to each other, without loss of generality, we only deal with “ $<$ ” in the sequel. In a monitoring scenario, as $\mathbf{x}$ is determined by an event stream λ , it is more reasonable to consider the attained values of $\mathbf{x}$ as a realisation of $\bar{\mathbf{X}}$, rather than the exact parameter values. Therefore, we should view the VOF f as a *transformation* from random sample means $\bar{\mathbf{X}}$ to a target random variable $Y = f(\bar{\mathbf{X}})$ and consider whether $Y < h$ is *statistically significant* after $\bar{\mathbf{X}}$ is elicited. More specifically, let $Y_{\text{true}} = f(E(\mathbf{X})) = f(E(\bar{\mathbf{X}}))$ be the true output value (where $E(\mathbf{X})$ denotes the expected value of $\mathbf{X}$), and $\mathbf{x}_0$ is a realisation of $\bar{\mathbf{X}}$ (after a sample is drawn); we have the following *statistical hypothesis*:

$$\begin{aligned}H_0 : Y_{\text{true}} < h & \quad (\text{null hypothesis}), \\ H_1 : Y_{\text{true}} \geq h & \quad (\text{alternative hypothesis}).\end{aligned}$$

Our objective is to determine the *p-value* given by

$$\Pr(Y - Y_{\text{true}} \geq |h - f(\mathbf{x}_0)|) \text{ if } H_0 \text{ is true.}$$

In statistics, *p-value* is a measurement of the unlikely output in the sense that a smaller *p-value* leads to a more statistically significant output. Moreover, *p-value* is inversely

Fig. 9. The QV4M framework

proportional to the size of sample data determining $\bar{X}$. In performance monitoring, we can use p -value to decide whether a significant output can be produced or should attain more data for the parameters. Specifically, if $1 - p$ is less than an acceptable *level of confidence* (say, 0.95), then the null hypothesis is rejected.

Notice that our aforementioned objective can be presented in an alternative (and equivalent) way: inferring a distance d from Y_{true} in the distribution of Y that gives rise to a specific p -value (say, $0.05 = 1 - 0.95$), and then comparing d and $|h - f(x_0)|$.

It is also noteworthy that by associating model parameters with random variables, we implicitly presume that the (unknown) true parameter values are fixed for sampling (i.e., during the event observation). In practice, even though the parameters are effected the environmental dynamics, we can determine a suitable sampling time frame and assume that the change of the true parameter values is negligible in this time frame.

4.3 Framework Overview

The design of the QV4M framework (Figure 9) is in response to the challenges discussed in Section 2.3. It infers the statistical significance of the monitoring output based on observed events and separates the online computation and the offline computation. In the offline phase, the input includes an ESS model and one or more property specifications. As mentioned, an ESS model is a parametric DTMC $\mathcal{M}_x$ and each property specification is formalised as a PRCTL formula Φ . For each model and property, the offline computation produces a VOF either in a closed form or as a matrix equation. On the online phase, the events are sampled and used to instantiate the parameters in the model. Then, the online computation provides verification results and their partial derivatives. Next, in order to determine whether a result satisfies (or violates) a prescribed requirement, a statistical method is employed to infer the statistical significance (i.e., a p -value). If it is significant, then a final monitoring output is reached; otherwise, it rolls

back to the sampling step to collect more sampling data. Finally, the monitoring is continuous and thus all the steps on the online phase are repeated periodically. The methods in QV4M are summarised in Table 1 and presented in detail in the next section.

5 COMPUTATION AND INFERENCE

In this section, we focus on the underlying technique of QV4M. In the first two subsections, we present the two computation methods (i.e., the matrix-iteration method and parametric model checking), which compute a value and partial derivatives for the VOF numerically and symbolically, respectively. As both methods are published in the previous works [17], [29], we only describe and illustrate the main steps with priority queue example. In the last subsection, we present a statistical inference method to infer a p -value for the VOF.

5.1 Matrix-Iteration Method

In probabilistic model checking, iterative computation is a most common technique. For our probabilistic model checking problem (as formalised in Section 3.2), an iteration method is used to solve a *matrix equation* that is generated based on a DTMC model and a PRCTL formula. For the ease of presentation, we introduce an $n \times 1$ vector y_x with $n = |S|$ such that

$$y_x(s) = \llbracket \Phi \rrbracket_s^{\mathcal{M}_x}(x), \quad s \in S \text{ and } x \in \mathcal{D}. \quad (1)$$

In words, $y_x(s)$ represents the (probability or reward) value of Φ at the corresponding state s of $\mathcal{M}_x$. Therefore, $y_x(s_0)$ equals to $f(x)$. The matrix equation related to a probabilistic model checking problem can be expressed as follows:

$$A_x \cdot y_x = b_x \quad (2)$$

where A_x is an $n \times n$ matrix and b_x is an $n \times 1$ vector. A_x and b_x can be constructed by using basic graph analysis and matrix manipulation. We remark that A_x is a non-singular matrix and thus its inverse matrix exists. Equation (2) can be solved by standard methods (e.g., the power method, Jacobi method and Gauss-Seidel method). A comprehensive reference to the technical details is found in the literature, e.g., [27]. Here we use our running example to illustrate the matrix equation. Recall that Φ_1 and Φ_2 are two metrics for the priority queue model. Figures 10a and 10c contains the matrix equations for model checking this model against Φ_1 and Φ_2 , respectively. Note that it is easy to change these two matrix equations into the form in Equation (2).

The fact that the VOF f can be expressed as Equation (2) also implies that f is smooth. Let x be a variable appearing in x . Based on Equation (2), we also have that

$$\frac{\partial A_x}{\partial x} \cdot y_x + A_x \cdot \frac{\partial y_x}{\partial x} = \frac{\partial b_x}{\partial x} \quad (3)$$

where $\frac{\partial y_x}{\partial x}(s_0)$ is $\frac{\partial f}{\partial x}$. Moreover, it is straightforward to generate $\frac{\partial A_x}{\partial x}$ and $\frac{\partial b_x}{\partial x}$ based on A_x and b_x by elementary differentiation. Therefore, Equation (3) can be solved by the same numerical algorithms that solve Equation (2). A detailed description of the power method for solving Equation (3) (for formulas without the reward operators) is presented by Su et al. [29]. The main steps and technique of this method are summarised in Table 1.

TABLE 1
Summary of main methods in QV4M

METHOD NAME	OFFLINE/ONLINE PHASE	TECHNIQUE
Matrix-iteration	Generation of parametric matrix equation (Offline and Online)	(Symbolic) differentiation, graph analysis, parametric matrix generation, and value substitution
	Matrix equation solving (Online)	Iterative computation (e.g., power method, Jacobi and Gauss-Seidel)
Parametric model checking	Generation of closed-form function (Offline)	State-elimination, (symbolic) differentiation
	Function valuation (Online)	Value substitution and elementary arithmetic
Statistical inference	(Online)	Multivariate- Δ method and first-order approximation

$$\begin{cases} \begin{bmatrix} y_{00} \\ y_{01} \\ y_{10} \\ y_{11} \end{bmatrix} = \begin{bmatrix} 0 & \frac{p}{p+q} & \frac{q}{p+q} & 0 \\ r & 0 & 0 & q \\ r & 0 & 0 & p \\ 0 & 0 & r & 0 \end{bmatrix} \cdot \begin{bmatrix} y_{00} \\ y_{01} \\ y_{10} \\ y_{11} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ q \\ q \end{bmatrix} \\ y_{02} = y_{12} = 0 \\ y_{20} = y_{21} = y_{22} = 1 \end{cases}$$

(a) Matrix equation for Φ_1

$$\frac{p^4 + qp^3 - p^3 + 2qp^2 + p^2}{p^4 + qp^3 - q^2p^2 - q^3p - p^3 + 2qp^2 + 3q^2p + p^2 + q^2}$$

(b) Closed-form expression for Φ_1

$$\begin{cases} \begin{bmatrix} z_{00} \\ z_{01} \\ z_{10} \\ z_{11} \end{bmatrix} = \begin{bmatrix} 0 & \frac{p}{p+q} & \frac{q}{p+q} & 0 \\ r & 0 & 0 & q \\ r & 0 & 0 & p \\ 0 & 0 & r & 0 \end{bmatrix} \cdot \begin{bmatrix} z_{00} \\ z_{01} \\ z_{10} \\ z_{11} \end{bmatrix} + \begin{bmatrix} 0 \\ pc \\ qc \\ (p+q)c \end{bmatrix} \\ z_{ij} = 0 \quad \text{if } i = 2 \text{ or } j = 2 \end{cases}$$

(c) Matrix equation for Φ_2

$$\frac{(2p^3 + 2qp^2 - 2p^2 + 2qp + 2p + 2q)c}{p^4 + qp^3 - q^2p^2 - q^3p - p^3 + 2qp^2 + 3q^2p + p^2 + q^2}$$

(d) Closed-form expression for Φ_2

Fig. 10. Matrix equations and closed-form expressions

5.2 Parametric Model Checking

Parametric model checking is a symbolic technique of probabilistic model checking. This technique exploits the fact that each RPCTL formula can be interpreted as a regular expression on the parametric DTMC; in other words, the closed-form expression of the VOF is a regular expression. Parametric model checking employs a state-elimination algorithm adopted from the theory of automata to compute the regular expression for a given parametric DTMC and a given RPCTL formula. This algorithm was originally proposed by Daws [9]. A detailed presentation of the algorithm (and its extension) is provided by Hahn et al. [17].

In our example, the closed-form expressions (denoted as f_1 and f_2) for Φ_1 and Φ_2 are presented in Figures 10b and 10d, respectively. Clearly, f_1 and f_2 are rational functions on p, q (which are fractions of polynomial functions on p, q). We emphasize that parametric model checking is *equivalent* to the matrix-iteration method in the following sense: For all p, q in their domain (i.e., $0 < p, q < 1$ and $p + q < 1$), $y_{00} = f_1(p, q)$ and $z_{00} = f_2(p, q)$.

Once the closed-form expression of the VOF f is obtained, it is straightforward to compute the closed-form expression of its partial derivatives $\frac{\partial f}{\partial x}$. For example, for the two functions in Figures 10b and 10d, partial differentiation

can produce their partial derivatives. Table 1 also includes a summary of the main steps and technique for parametric model checking.

5.3 Multivariate- Δ Statistical Inference

Recall that our objective is to infer a p -value from a given distance d or, equivalently, a distance d from a given p -value. To achieve this, one straightforward solution is resorting to the distribution (i.e., cumulative distribution function (CDF)) of $Y = f(\bar{\mathbf{X}})$. But because the joint CDF of $\bar{\mathbf{X}}$ is unknown and f is usually non-linear and complex, it is impractical to pursue the CFD of Y . However, a well-known fact in statistics is that, as the sample size (i.e., length of topic streams) increases, Y is approximately normally distributed. By exploiting the approximate normal distribution $\mathcal{N}(Y_{\text{true}}, \delta^2)$ of Y (with a known δ^2), we can infer either a p -value or its corresponding distance d approximately. Therefore, we aim at inferring δ^2 .

Based on (one of) the previous two methods, our framework QV4M employs a statistical inference method called the *multivariate- Δ method* to infer a p -value. For convenience, we use $\boldsymbol{\mu} = (\mu_1, \dots, \mu_m)$ to denote $E(\mathbf{X})$ (i.e., the true values of $\mathbf{X}$) and let

$$\mathbf{d}(\mathbf{x}) = \left(\frac{\partial f(\mathbf{x})}{\partial x_1}, \dots, \frac{\partial f(\mathbf{x})}{\partial x_m} \right).$$

In the following, we employ the statistical concepts *convergence in distribution* and *consistent estimator*, which can be found in the textbooks in statistics (e.g., [19, Ch. 4]). The following theorem in the multivariate Δ -method, which is a special case of [19, Theorem 4.5.6].

Proposition 1. *Let ι be the sample size of $\mathbf{X}$ and $\boldsymbol{\Sigma}$ be the covariance matrix of $\mathbf{X}$. If $\mathbf{d}(\boldsymbol{\mu})\boldsymbol{\Sigma}\mathbf{d}(\boldsymbol{\mu})^T \neq 0$ then, as ι increases, the random variable*

$$\iota^{\frac{1}{2}}(Y - Y_{\text{true}}) \quad (4)$$

converges to $N(0, \mathbf{d}(\boldsymbol{\mu})\boldsymbol{\Sigma}\mathbf{d}(\boldsymbol{\mu})^T)$ in distribution.

Let $\boldsymbol{\Sigma}_0$ be the sample covariance matrix of $\bar{\mathbf{X}}$ based on the stream λ . Thus, $\boldsymbol{\Sigma}_0$ is a consistent estimator of $\boldsymbol{\Sigma}$. Because $\bar{\mathbf{X}}$ is a consistent estimator of $\boldsymbol{\mu}$ and $\mathbf{d}$ is continuous, $\mathbf{d}(\bar{\mathbf{X}})$ is a consistent estimator of $\mathbf{d}(\boldsymbol{\mu})$. According to Proposition 1, in practice, once we draw a concrete sample mean for $\bar{\mathbf{X}}$ such that $\mathbf{d}(\bar{\mathbf{X}})\boldsymbol{\Sigma}_0\mathbf{d}(\bar{\mathbf{X}})^T \neq 0$ (which, given the condition of the theorem, is likely so for a large ι), we can view Y as approximately normally distributed with distribution $N(Y_{\text{true}}, \delta_Y^2)$ where

$$\delta_Y^2 = \iota^{-1}\mathbf{d}(\bar{\mathbf{X}})\boldsymbol{\Sigma}_0\mathbf{d}(\bar{\mathbf{X}})^T. \quad (5)$$

Although in practice μ is unknown and so is Y_{true} (which equals to $f(\mu)$), an important gain of Proposition 1 is an approximate normal distribution $N(Y_{\text{true}}, \delta_Y^2)$ of Y where δ_Y^2 can be estimated. Suppose a concrete sample mean for $\bar{\mathbf{X}}$ is drawn and the threshold h is given. With this approximate normal distribution of Y , based on the p -value definition in Section 4.2, the inference of a p -value is straightforward.

Computational complexity and convergence rate. We present the time complexity of our two computational methods and the convergence rate of our statistical inference. The matrix-iteration method requires to compute the solutions to Equation (3) for each parameter. The time of solving Equation (3) directly (e.g., by using Gaussian elimination) is $O(|\mathcal{M}|^3)$ where $\mathcal{M}$ is the DTMC model. Thus, the overall time complexity of this method is $O(|\mathcal{M}|^3 m)$ where m is the number of model parameters. However, to solve large-size equations, iterative computation (e.g., by using the power, Jacobi and Gauss-Seidel methods) is usually more efficient in practice, even though it might take exponentially many iterations to reach the fixed point in the worst case. For parametric model checking, since the state-elimination algorithm can also be done in time $O(|\mathcal{M}|^3)$, the overall time complexity is also $O(|\mathcal{M}|^3 m)$. However, the size of the closed-form expression is $|\mathcal{M}|^{O(\log |\mathcal{M}|)}$ in the worst case, which is usually the bottleneck of parametric model checking in practice [17]. For the multivariate Δ -method, the convergence rate is $O(1/\sqrt{\iota})$ where ι is the sample size. In other words, in order to obtain an accuracy $O(1/\sqrt{\iota})$ in our statistical inference, the data cost (i.e., the number of samples) is $O(\iota)$. In Section 7, we present an empirical evaluation of the computation time and the data cost.

6 MULTI-TOPIC STREAM ANALYSIS

Real-world streaming pipeline is complex and may comprise multiple topics. One important question is whether random variables corresponding to events across different topics *can* (or *should*) be sampled either separately or together. If the stream is considered as a single random sample regardless of the topic multiplicity, we can apply the multivariate Δ -method (which relies on Assumption 1). However, if only events belonging to the same topic constitute a random sample, the multivariate Δ -method is not applicable. In view of this, we present a multi-topic stream model and an alternative statistical inference method to infer a p -value that relies on different assumptions.

Let $\mathcal{I}$ be a partition on $\{1, \dots, m\}$ where $m = |\mathbf{x}|$. A *topic* is a subset of events $\mathcal{E}_I = \{e_i \in \mathcal{E} \mid i \in I\}$ where $I \in \mathcal{I}$. A *topic stream* is $\lambda_I \in \mathcal{E}_I^*$. Let $\mathbf{X}_I = (X_i)_{i \in I}$ be a (sub-)vector of Bernoulli random variables. As we intend to sample a subset of parameters for each topic stream, we assume the following proposition:

Assumption 2. For each $I \in \mathcal{I}$, the topic stream λ_I is a random sample of $\mathbf{X}_I$.

It is easy to see that Assumption 2 can be derived from Assumption 1. Thus, Assumption 1 is a more restricted assumption. However, if random variables for different topics are sampled independently, in order to infer a p -value, we need the following additional assumption:

Assumption 3. $\mathbf{X}_I$ and $\mathbf{X}_{I'}$ are independent if $I \neq I'$.

The first-order approximation in the Taylor expansion of f is as follows:

$$f(\mathbf{x}) \approx g(\mathbf{x}) = f(\mu) + \sum_{i=1}^m (x_i - \mu_i) f'_i(\mu). \quad (6)$$

For two random variables X, X' , let $\text{cov}(X, X')$ denote their *covariance*. Note that if X and X' are independent then $\text{cov}(X, X') = 0$, and that $\text{cov}(X, X) = \text{var}(X)$.

Proposition 2. Let ι_I is the sample size of $\mathbf{x}_I$ and

$$\text{var}_I(Y) = \sum_{i,j \in I} \text{cov}(X_i, X_j) f'_i(\mu) f'_j(\mu). \quad (7)$$

Then,

$$\text{var}(Y) \approx \sum_{I \in \mathcal{I}} \text{var}_I(Y) / \iota_I. \quad (8)$$

Proof. By the linear approximation in Equation (6),

$$\begin{aligned} \text{var}(Y) &\approx \text{var}(g(\bar{\mathbf{X}})) = \text{var}\left(\sum_{i=1}^k (\bar{X}_i - \mu_i) f'_i(\mu)\right) \\ &\text{(by definition)} = \sum_{i,j=1}^k \text{cov}(\bar{X}_i, \bar{X}_j) f'_i(\mu) f'_j(\mu). \end{aligned}$$

By the second interpretation, if $i \in I, j \in I'$ and $I \neq I'$, then $\text{cov}(X_i, X_j) = 0$ and so $\text{cov}(\bar{X}_i, \bar{X}_j) = 0$. Also, if $i, j \in I$ then $\text{cov}(X_i, X_j) / \iota_I = \text{cov}(\bar{X}_i, \bar{X}_j)$ where ι_I is the sample size. The proposition follows. $\square$

Proposition 3. For each $I \in \mathcal{I}$, as ι_I increases,

$$\sigma_I^2 = \sum_{i,j \in I} \text{cov}(X_i, X_j) f'_i(\bar{\mathbf{X}}) f'_j(\bar{\mathbf{X}}) \quad (9)$$

converges to $\text{var}_I(Y)$ in distribution.

Proof. For each $I \in \mathcal{I}$ and $i, j \in I$, as ι_I increases, $f'_i(\bar{\mathbf{X}})$ and $f'_j(\bar{\mathbf{X}})$ converge to $f'_i(\mu)$ and $f'_j(\mu)$ respectively, and thus $f'_i(\bar{\mathbf{X}}) f'_j(\bar{\mathbf{X}})$ converges to $f'_i(\mu) f'_j(\mu)$. The proposition follows immediately. $\square$

By Propositions 2 and 3, σ_I^2 / ι_I is an estimator of $\text{var}_I(Y)$ for each $I \in \mathcal{I}$. Therefore, the following quantity is an approximation of $\text{var}(Y)$:

$$\sigma_Y^2 = \sum_{I \in \mathcal{I}} \sigma_I^2 / \iota_I. \quad (10)$$

To infer a p -value, as before we resort to an approximate distribution of Y . According to the Central Limit Theorem, each $\bar{\mathbf{X}}_I$ in $\bar{\mathbf{X}}$ is approximately normally distributed. Also, Assumption 3 says that $\bar{\mathbf{X}}_I$ is independent of other $\bar{\mathbf{X}}_{I'}$ for $I' \in \mathcal{I} \setminus \{I\}$. Since g is a linear transform of $\bar{\mathbf{X}}$, we conclude that $g(\bar{\mathbf{X}})$ is approximately normal distributed. Therefore, it is reasonable to assume that Y is approximately normally distributed and has the distribution $N(Y_{\text{true}}, \sigma_Y^2)$. Then, a p -value can be inferred based on $N(Y_{\text{true}}, \sigma_Y^2)$ as before.

We present a brief comparison between the multivariate Δ -method and the first-order approximation inference. First of all, if Assumptions 1 and 3 (and thus Assumption 2) are assumed, the two inference methods are equivalent, namely, producing the same estimate for the variance of $Y = f(\mathbf{X})$. But these two methods have different strength in application. The multivariate Δ -method must draw random samples across stream topics but is applicable to correlated or independent random variables. With the independence assumption (i.e., Assumption 3), the first-order approximation inference allows more flexible, topic-wise sampling. We

TABLE 2
Model summary for priority queue (PQ), “kitchensink” workflow (KS)
and HTTP request handling (RH)

Model	# Param.	Complex Expr.	Loop	Incremental Size
PQ	2	fractional	yes	yes (on max #task)
KS	12	no	yes	yes (on max #retry)
RH	13	no	no	no (fix-sized)

also note that the convergence rate of the first-order approximation inference is the same as that of the multivariate Δ -method, namely $O(1/\sqrt{\iota_I})$ where $\iota_I = \min_{I \in \mathcal{I}} \iota_I$ (i.e., the smallest sample size for topic streams).

7 CASE STUDIES

In the case studies, we aim to evaluate the following two aspects of QV4M:

- *Computation time* (Section 7.1): The time of offline and online computation in QV4M with respect to the model size, which does *not* include the sampling time.
- *Data cost* (Section 7.2): The convergence of statistical inference in QV4M with respect to the sample size.

We include two example models (i.e., a microservice workflow model and an HTTP request handling model), in addition to the priority queue model in our running example. This can further validate the ability of our framework to analyse various useful ESS models and performance metrics. The model specification files and other supporting files for the two case studies are available in the first author’s homepage.²

7.1 Computation Time Evaluation

As mentioned, the events in our microservice orchestration system (Figure 3) can be used to create different streams in a Kafka pipeline. We have considered the queuing event stream in our running example, but we can model and analyse various other event streams. Typically, the Workflow Server in the system stores and manages multiple predefined workflow blueprints. A (possibly large) number of instances of the same or different workflow blueprints may run concurrently and be tracked by the Workflow Server. For each workflow, we can analyse its execution metrics based on the lifecycle events of its instances. This justifies the modelling variability of QV4M.

In this case study, we used the priority queue model and a “kitchensink” microservice workflow³. We assumed that the tasks and sub-workflows in the “kitchensink” workflow may fail and, if that happens, a fixed number of retries can be made. Table 2 presents a summary of the two models. The computation time that we evaluated does *not* include the time related to collecting and aggregating events. As the time of statistical inference (by the multivariate Δ -method or first-order approximation method) is almost instant, we only considered the offline and online computation time (c.f., Figure 9).

Table 3 includes the evaluation result of computation time (i.e., elapsed time) by using matrix-iteration method

TABLE 3
Evaluation results of computation time (in seconds)

PQ			Q.V.	P.M.C.		M.I.	
max #task	#state	#tran		offline	online	offline	online
5	36	95	2.9	2	<0.1	<0.5	<0.5
6	49	132	2.9	15	<0.1	<0.5	<0.5
7	64	175	2.9	48	<0.1	0.5	0.5
8	81	224	2.9	372	<0.1	0.7	0.8
9	100	279	2.9	-	-	1.2	1.3
20	441	1228	2.9	-	-	2.1	3.0
50	2601	7700	3.0	-	-	15.0	14.6
100	10201	30400	5.6	-	-	84.5	80.2

KS			Q.V.	P.M.C.		M.I.	
max #retry	#state	#tran		offline	online	offline	online
5	309	918	2.9	<1.0	<0.1	5	0.5
7	487	1546	2.9	3.6	<0.1	9.5	0.8
9	705	2334	2.9	19.2	<0.1	12	1
11	963	3282	2.9	54	<0.1	17	1.5
13	1261	4390	2.9	144	<0.1	24	2
15	1599	5658	2.9	364	<0.1	30	3
17	1977	7086	2.9	-	-	38	4
20	2619	9528	2.9	-	-	51	6.5
50	13989	53748	3.8	-	-	289	126

(i.e., the “M.I.” column) and parametric model checking (i.e., the “P.M.C.” column), where the “-” flag in the table indicates an out-of-memory error. For the matrix-iteration method, we developed a self-implemented prototype in Python; for parametric model checking, we employed the parametric model checking tool PROPhESY [10]. The experiment results in Table 3 are based on the two models with different model size. For the priority queue model, we analysed the temporal property Φ_1 . For the “kitchensink” workflow model, we analysed the probability of reaching a successful state. The time for online computation in Table 3 is just for one execution (event though the online phase in QV4M may contain a loop). For reference purposes, the table also includes the time of standard quantitative verification (i.e., the “Q.V.” column) by using the tool PRISM. The experiment environment is a Linux VM with a 2.8GHz CPU core and 2GB memory.

The experimental data shows that the matrix-iteration method is more scalable than parametric model checking in general. The major bottleneck of parametric model checking is the offline stage. But if a closed-form function can be pre-computed, the online stage of parametric model checking is almost instant. The most time-consuming part in our self-implemented matrix-iteration method is the computation related to symbolic parameter expressions (c.f., Table 1), while the actual numerical matrix-iteration is very efficient. But the runtime of both methods is much worse runtime than that of standard quantitative verification. (It appeared that the elapsed time in the “Q.V.” column is mostly the startup time of calling PRISM in a Linux shell.) Overall, our implementation of QV4M has reasonable computational efficiency to analyse ESS models in realistic size and with non-trivial parametric structures.

7.2 Data Cost Evaluation

HTTP request-handling model. In this case study, besides the priority queue and workflow models, we included an ESS

2. <https://documents.uow.edu.au/~guoxin/QV4M/>
3. <https://netflix.github.io/conductor/labs/kitchensink/>

TABLE 4
HTTP request-handling performance metrics

PM	Description	PRCTL formula
PM1	Probability of handling a request without accessing the database or file server (i.e., cache hit probability)	$P_{=?}[\neg(\text{Database} \wedge \text{FileServer}) \cup \text{HttpResponse}]$
PM2	Probability of successfully handling a request	$P_{=?}[\text{F } \text{HttpResponse}]$
PM3	Probability of processing a request in 4 operations	$P_{=?}[\text{F}^{\leq 4} \text{HttpResponse}]$
PM4	Average cost of handling a request:	$C_{=?}^c[\text{HttpResponse} \wedge \text{TooManyConnections} \wedge \text{ServerUnavailable}]$
PM5	Average time of handling a request	$C_{=?}^t[\text{HttpResponse} \wedge \text{TooManyConnections} \wedge \text{ServerUnavailable}]$

TABLE 5
Evaluation results of data cost

	Prob. Output	Values of d	Scaling Factor	Sample Size					
				2,000	5,000	10,000	20,000	50,000	100,000
PQ	0.9925	inferred reference	10^{-3}	18.23 ± 4.96 17.27	11.69 ± 1.89 10.39	7.960 ± 0.964 6.947	5.543 ± 0.313 5.306	3.553 ± 0.149 3.296	2.520 ± 0.035 2.507
KS	0.9423	inferred reference	10^{-4}	8.223 ± 0.541 9.382	5.033 ± 0.176 5.618	3.555 ± 0.130 3.949	2.506 ± 0.050 2.735	1.577 ± 0.029 1.681	1.117 ± 0.010 1.185
RH PM1	0.7004	inferred reference	10^{-3}	19.53 ± 0.55 18.95	12.35 ± 0.21 12.23	8.735 ± 0.120 8.703	6.188 ± 0.054 6.286	3.910 ± 0.027 3.999	2.763 ± 0.012 2.664
RH PM2	0.9992	inferred reference	10^{-4}	10.26 ± 7.29 13.24	6.459 ± 2.286 7.985	4.779 ± 1.566 5.584	3.415 ± 0.761 3.672	2.107 ± 0.307 2.244	1.489 ± 0.149 1.554
RH PM3	0.9169	inferred reference	10^{-3}	7.003 ± 0.231 7.209	4.456 ± 0.108 4.360	3.141 ± 0.052 3.132	2.226 ± 0.024 2.281	1.403 ± 0.010 1.411	0.992 ± 0.005 0.999
RH PM4	2.292	inferred reference	10^{-2}	6.463 ± 0.118 6.441	4.102 ± 0.034 4.225	2.897 ± 0.024 2.925	2.053 ± 0.008 2.105	1.297 ± 0.003 1.329	0.916 ± 0.003 0.962
RH PM5	13.42	inferred reference	10^{-1}	8.165 ± 0.219 7.920	5.236 ± 0.096 5.041	3.720 ± 0.044 3.520	2.640 ± 0.023 2.518	1.680 ± 0.018 1.574	1.189 ± 0.009 1.143

Fig. 11. Model of HTTP request handling in a Web application

model of the HTTP request-handling process (Figure 11) (which is an example taken from [3]). This model represents a Web application consisting of an HTTP proxy server, a Web server and an application server. To serve client requests, the application accesses structured data and static content (e.g., text files and images) stored in a database and on a file server, respectively. Both types of contents are cached by cache servers. We analysed five behavioural performance metrics (PM1–PM5 in Table 4) for the Web application, which are also taken from [3]. In this model, some transition probabilities are pre-determined and others are parameters that have to be estimated by sampling the events. The two rewards assigned to each state represent the cost and time of staying at the state, respectively. We formalised PM1–PM5 as PRCTL formulas (also in Table 4) and pre-computed their closed-form expressions by parametric model checking.

We treated the probability parameters of in all three model as the expected values of Bernoulli random distributions and used simulation to create random samples for the probability parameters. Recall that, given a set of samples for the parameters, a distance d can be inferred from a p -value (c.f., Section 4.2). In this experiment, we fixed the p -value as 0.005 and inferred the distance d (c.f., Section 4.2). By repeating the sampling process for each selected sample size (which is between 2,000 to 100,000) for each parameter, we inferred a set of values for d in each case. We selected the inferred values between 5th percentile and 95th percentile and compared those *inferred values* of d with a *reference value* of d , which is simulated by a Monte Carlo method separately. Table 5 summarise our experiment results, which demonstrate a reasonable convergence rate of our statistical inference methods with respect to the sample size. We observed that, as the sample size increases, the inferred ranges of d are narrowed and the mean values of those ranges are closer to the reference value of d for all models and performance metrics. We also found that the convergence rate is more sensitive to the number of parameters and the extremeness of the probabilistic output in the “Prob. Output” column (e.g., how close the output is close to 0 or 1) than other factors (such as the existence of complex parametric expressions and loops).

We present one remark for our empirical evaluation. Even though both cases of the evaluation manifest increasing analysis quality with increasing sample size, in many real-world situations of performance monitoring, we do not endeavour to collect an as-large-as-possible sample. Some-

times, even with a small sample, the inferred distance d is much smaller than the difference between the output value and the threshold, we still produce a statistically significant output. Also, in practice we need to take factors such as the timeliness of data and the sampling cost into consideration.

8 RELATED WORK

Probabilistic model checking as a formal verification technique for stochastic systems is matured over the past two decades. Katoen [22] presented a comprehensive review on this technique as well as its applications. We report the recent development in probabilistic model checking that is most relevant to the computation methods of QV4M.

Parametric model checking was first introduced by Daws [9], which employs a state-elimination algorithm from the theory of automata to compute the exact value (i.e., no truncation) for a DTMC verification problem. This method can also handle DTMC in the presence of undetermined parameters. Hahn et al. [17] improved and extended Daws's method to cope with the verification of a parametric DTMC with parametric rewards and parametric Markov Decision Processes (MDPs). Jansen et al. [20] further escalated the efficiency of this method by using graph decomposition and polynomial factorisation. A different method of parametric model checking based on the Gauss-Jordan elimination directly is employed by Filieri et al. [15] in their runtime quantitative verification framework. It is noteworthy that parametric model checking is closely related to the problem of parameter synthesis. In view of the computational cost of the state-elimination algorithm, some research [25] deals with the parameter synthesis problem by getting around the closed-form verification functions.

Asymptotic perturbation analysis [6], [29] extends the standard matrix-iteration methods (e.g., the power method, Jacobi method and Gauss-Seidel method) in probabilistic model checking to compute the partial derivatives of a verification problem. This approach aims to estimate an accurate worst-case bound for the verification output when the model parameters are subject to small perturbations. Compared with parametric model checking, the matrix-iteration method can be applied to a verification problem resulting in a non-rational function (i.e., the time-bounded model checking for continuous-time Markov chains [28]).

Calinescu et al. [3] proposed the framework FACT, which is the first technique that supports the formal verification of DTMC that exploits confidence intervals. FACT employs parametric model checking to compute a closed-form verification function. Then, it samples the parameters and infers their confidence levels and confidence intervals. Last, it implements a hill-climbing algorithm to compute a best confidence interval whose confidence level is the product of confidence levels of parameters. Each hill-climbing iteration invokes a non-linear programming solver to compute one "candidate" confidence interval for the verification output from the given confidence intervals of parameters. Therefore, FACT is computationally expensive. By contrast, our framework QV4M uses a light-weight statistical inference method and adopts a matrix-iteration method as a complement to parametric model checking.

There is extensive literature on formal specification of and reasoning about QoS properties of software systems (e.g., [2], [26], [32]). Calinescu et al. [4] presented a systematic and comprehensive framework QoS MOS based on probabilistic model checking for QoS management and optimisation for service-based systems. QoS MOS integrates a suite of tools that support all four run-time stages in system self-adaptation, namely monitoring, analysing, planning and executing (MAPE). Ghezzi et al. [16] proposed a model-driven framework ADAM to aid the adaptation to non-functional manifestations of uncertainty. ADAM consists of a generator, which generates a probabilistic model from the UML Activity Diagram, and an interpreter, which searches for an execution path in the model that satisfies the non-functional requirements. The runtime quantitative verification framework proposed by Filieri et al. [15] also aims to support the analysing stage in the system self-adaptation.

There are Bayesian methods for learning parameters in a parametric DTMC from the historical data [5], [14]. These parameter learning methods are complementary to our framework which employs simple point estimates for sample means and sample covariances of the parameters. Finally, we mention the simulation-based approaches to probabilistic verification [18], [33]. In those approaches, the parameter sampling results in a set of paths of the Markov model. This is in sharp contrast to our data distribution assumption and that in FACT.

9 CONCLUSIONS

We have presented QV4M, which is a performance monitoring framework for ESS systems based on probabilistic model checking. QV4M includes a parametric DTMC model with transition probabilities, which are characterised by rational parametric expressions, and an event stream model, which associates the parameters with random samples of events. Unlike standard probabilistic model checking, QV4M produces not only a probabilistic verification output but also the statistical significance of this output. QV4M uses a matrix-iteration method or parametric model checking to compute the partial derivatives of a quantitative verification output. Our empirical evaluation has demonstrated its practicality.

For the future work, one direction is to establish a more systemised framework for QV4M. For example, we can incorporate a feedback loop into QV4M to determine how much more sample data should be collected if the output is not statistically significant. A second direction is to enhance the model parameter estimation with more sophisticated learning methods. Last, it is also interesting to validate the effect of the parameter correlation on the statistical inference accuracy in QV4M.

ACKNOWLEDGEMENTS

This work was partially supported by the National Major Science and Technology Projects of China under grants 2018AAA0100703 and 2018AAA0100700, the National Natural Science Foundation of China under grants 61502260 and 61977012, and the Singapore Ministry of Education under grant MOE2015-T2-1-137.

REFERENCES

- [1] C. Baier and J.-P. Katoen, *Principles of Model Checking*. The MIT Press, 2008.
- [2] A. Benveniste, C. Jard, A. Kattepur, S. Rosario, and J. Thywissen, "QoS-aware management of monotonic service orchestrations," *Formal Methods in System Design*, vol. 44, no. 1, pp. 1–43, 2014.
- [3] R. Calinescu, C. Ghezzi, K. Johnson, M. Pezz, 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, March 2016.
- [4] 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.
- [5] R. Calinescu and M. Kwiatkowska, "Using quantitative analysis to implement autonomic IT systems," in *Proceedings of the 31st International Conference on Software Engineering*, ser. ICSE'09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 100–110.
- [6] 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.
- [7] S.-W. Cheng, D. Garlan, and B. Schmerl, "Architecture-based self-adaptation in the presence of multiple objectives," in *ICSE 2006 Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, Shanghai, China, 21–22 2006.
- [8] V. Cortellessa and V. Grassi, "A performance-based methodology to early evaluate the effectiveness of mobile software architectures," *J. Log. Algebr. Program.*, vol. 51, no. 1, pp. 77–100, 2002.
- [9] 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.
- [10] C. Dehnert, S. Junges, N. Jansen, F. Corzilius, M. Volk, H. Buntjes, J.-P. Katoen, and E. Ábrahám, "Prophesy: A probabilistic parameter synthesis tool," in *Computer Aided Verification*, D. Kroening and C. S. Păsăreanu, Eds. Cham: Springer International Publishing, 2015, pp. 214–231.
- [11] C. Dehnert, S. Junges, J. Katoen, and M. Volk, "A storm is coming: A modern probabilistic model checker," *CoRR*, vol. abs/1702.04311, 2017. [Online]. Available: <http://arxiv.org/abs/1702.04311>
- [12] T. Dunning and E. Friedman, *Streaming architecture: new designs using Apache Kafka and MapR streams*. O'Reilly Media, Inc., 2016.
- [13] L. Etxeberria, C. Trubiani, V. Cortellessa, and G. Sagardui, "Performance-based selection of software and hardware features under parameter uncertainty," in *Proceedings of the 10th International ACM Sigsoft Conference on Quality of Software Architectures*, ser. QoSA '14. New York, NY, USA: ACM, 2014, pp. 23–32.
- [14] A. Filieri, L. Grunske, and A. Leva, "Lightweight adaptive filtering for efficient learning and updating of probabilistic models," in *Proceedings of the 37th International Conference on Software Engineering*, ser. ICSE'15. IEEE, 2015.
- [15] 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.
- [16] 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.
- [17] 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.
- [18] T. Héruault, R. Lassaigne, F. Magniette, and S. Peyronnet, "Approximate probabilistic model checking," in *Verification, Model Checking, and Abstract Interpretation*. Springer, 2004, pp. 73–84.
- [19] R. V. Hogg, J. W. McKean, and A. T. Craig, *Introduction to Mathematical Statistics* (6th ed). Pearson Prentice Hall, 2005.
- [20] N. Jansen, F. Corzilius, M. Volk, R. Wimmer, E. Ábrahám, J.-P. Katoen, and B. Becker, "Accelerating parametric probabilistic verification," in *Quantitative Evaluation of Systems*, G. Norman and W. Sanders, Eds. Cham: Springer International Publishing, 2014, pp. 404–420.
- [21] 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.
- [22] —, "The probabilistic model checking landscape," in *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*, ser. LICS '16. New York, NY, USA: ACM, 2016, pp. 31–45.
- [23] 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.
- [24] Y. R. S. Llerena, G. Su, and D. S. Rosenblum, "Probabilistic model checking of perturbed mdps with applications to cloud computing," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2017. New York, NY, USA: ACM, 2017, pp. 454–464. [Online]. Available: <http://doi.acm.org/10.1145/3106237.3106301>
- [25] T. Quatmann, C. Dehnert, N. Jansen, S. Junges, and J. Katoen, "Parameter synthesis for markov models: Faster than ever," in *Automated Technology for Verification and Analysis - 14th International Symposium, ATVA 2016, Chiba, Japan, October 17–20, 2016, Proceedings*, 2016, pp. 50–67.
- [26] G. Rodrigues, D. Rosenblum, and S. Uchitel, "Using scenarios to predict the reliability of concurrent component-based software systems," in *Proceedings of the 8th International Conference on Fundamental Approaches to Software Engineering*, ser. FASE'05. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 111–126.
- [27] W. J. Stewart, *Introduction to the numerical solutions of Markov chains*. Princeton Univ. Press, 1994.
- [28] G. Su, T. Chen, Y. Feng, and D. S. Rosenblum, "Proeva: Run-time proactive performance evaluation based on continuous-time markov chains," in *Proceedings of the 39th International Conference on Software Engineering*, ser. ICSE '17. Piscataway, NJ, USA: IEEE Press, 2017, pp. 484–495.
- [29] G. Su, Y. Feng, T. Chen, and D. S. Rosenblum, "Asymptotic perturbation bounds for probabilistic model checking with empirically determined probability parameters," *IEEE Transactions on Software Engineering*, vol. 42, no. 7, pp. 623–639, 2016.
- [30] G. Su, D. S. Rosenblum, and G. Tamburrelli, "Reliability of run-time Quality-of-Service evaluation using parametric model checking," in *Proceedings of the 38th International Conference on Software Engineering*, ser. ICSE'16. New York, NY, USA: ACM, 2016.
- [31] J. Sun, Y. Liu, J. S. Dong, and J. Pang, "Pat: Towards flexible verification under fairness," in *Proceedings of the 21th International Conference on Computer Aided Verification (CAV'09)*, ser. Lecture Notes in Computer Science, vol. 5643. Springer, 2009, pp. 709–714.
- [32] A. Tarasyuk, E. Troubitsyna, and L. Laibinis, "Formal modelling and verification of service-oriented systems in probabilistic event-b," in *Integrated Formal Methods - 9th International Conference, IFM 2012, Pisa, Italy, June 18–21, 2012*, ser. Lecture Notes in Computer Science, J. Derrick, S. Gnesi, D. Latella, and H. Trehan, Eds., vol. 7321. Springer, 2012, pp. 237–252.
- [33] H. L. S. Younes and R. G. Simmons, "Statistical probabilistic model checking with a focus on time-bounded properties," *Information and Computation*, vol. 204, no. 9, pp. 1368–1409, 2006.

Guoxin Su is a lecturer at the University of Wollongong (UOW), Australia. He received his Ph.D. from the University of Technology Sydney in 2013. Before joining UOW, he was a research fellow and a senior research fellow with the School of Computing at the National University of Singapore. His research areas include software engineering, formal methods and big data. He is particularly interested in developing and applying novel probabilistic and learning methods for a variety of real-world software design and analysis problems. He has published a series of papers on the top-ranked journals and conferences in software engineering.

Li Liu is currently an associate professor at Chongqing University. He received his Ph.D. in Computer Science from the Université Paris-sud XI in 2008. He had served as an associate professor at Lanzhou University in China and also a Senior Research Fellow of School of Computing at the National University of Singapore. His research interests are in data analysis, pattern recognition and their applications on human behaviours. He aims to contribute in interdisciplinary research of computer science and human related disciplines. He has published widely in conferences and journals with more than 100 peer-reviewed publications. He has been the Principal Investigator of several funded projects from government and industry.

Minjie Zhang received her Bachelor of Computer Science degree from Fudan University, China in 1982 and Ph.D. degree from the University of New England, Australia in 1996. Dr Minjie Zhang is a full professor, the Director of Centre for Big Data Analytics and Intelligent Systems, and the Chair of the Research Cluster in Computer Science and IT, at the University of Wollongong (UOW), Australia. She is the author, or co-author, of over 250 research papers. Her research interests include multi-agent systems and their applications in complex domains, distributed artificial intelligence, smart modeling and simulation in complex systems, agent-based grid/cloud computing, and smart grid systems.

David S. Rosenblum is Provost's Chair Professor of Computer Science at the National University of Singapore (NUS) and Director of the NUS-Singtel Cyber Security Research & Development Laboratory. He received his PhD from Stanford University and joined NUS in April 2011, after holding positions as Research Scientist at AT&T Bell Labs (Murray Hill); Associate Professor at the University of California, Irvine; Principal Architect and Chief Technology Officer of PreCache (a technology startup funded by Sony Music); and Professor of Software Systems at University College London. His research interests span many problems in formal methods, software engineering, distributed systems, ubiquitous computing and machine learning. He is a Fellow of the ACM and IEEE.