Runtime Model Checking for SLA Compliance Monitoring and QoS Prediction

Giuseppe Cicotti, Luigi Coppolino*, Salvatore D'Antonio, Luigi Romano
University of Naples Parthenope, 80143 Napoli, Italy
{giuseppe.cicotti,luigi.coppolino,salvatore.dantonio,lrom}@uniparthneope.it

Abstract

Sophisticated workflows, where multiple parties cooperate towards the achievement of a shared goal are today common. In a market-oriented setup, it is key that effective mechanisms be available for providing accountability within the business process. The challenge is to be able to continuously monitor the progress of the business process, ideally,anticipating contract breaches and triggering corrective actions. In this paper we propose a novel QoS prediction approach which combines runtime monitoring of the real system with probabilistic model-checking on a parametric system model. To cope with the huge amount of data generated by the monitored system, while ensuring that parameters are extracted in a timing fashion, we relied on big data analytics solutions. To validate the proposed approach, a prototype of the QoS prediction framework has been developed, and an experimental campaign has been conducted with respect to a case study in the field of Smart Grids.

Keywords: Big Data Analytics, QoS Prediction, Model Checking, SLA compliance monitoring

1 Introduction

The service-oriented computing paradigm has been changing the way of creating, developing, and delivering new services and is the foundation of the Utility Computing service provisioning model, where a service provider repackages computing resources and infrastructure management, makes them available to the customer as needed, and charges for their specific usage. Utility Computing paved the way to the as a Service (aaS) model [27], that further propagated the idea of providing computing, application, and network resources as a metered service. One of the main advantages of the aaS model is the dramatic reduction of time, effort, and cost associated with the development of new services, since these can be created by integrating and reusing existing ones, including third party applications and legacy systems. This lowering of the bar has ultimately resulted in new services of an ever increasing complexity being created at a fast pace. The orchestration of services currently being developed reflects sophisticated workflows [13, 5], where multiple parties individually providing relatively simple services cooperate towards the achievement of a common goal, i.e. building a service with more functionalities and/or better quality. Obviously enough, this goal can only be reached if all parties that are in the critical paths of the workflow deliver what they promise, both in terms of functions and of quality. Failure to do so results in substantial detriments to the composite service, and possibly to its complete failure. In a market-oriented setup, it is thus key that effective mechanisms be available for providing accountability within the business process. With respect to quality aspects, two are the fundamental tools for specifying the terms of a service, namely Service Level Agreements (SLAs)[3, 12] and Operational Level Agreements (OLAs). An SLA is an agreement between two or more parties, where one is the customer

Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications, volume: 6, number: 2 (June), pp. 1-17

*Corresponding author: Department of Engineering, University of Naples Parthenope, Centro Direzionale Di Napoli Is. C4, 80143 Napoli, Italy, Tel: +39-081-547-6702

and the others are service providers. OLAs, i.e. contracts between the service provider and other third parties, may be used by internal groups to support SLAs. The challenge is to be able to continuously monitor the progress of the business process, and to timely spot breaches of quality contracts, i.e. situations where one or more parties fail to meet the agreed upon QoS levels [6, 11]. Ideally, one should be able to anticipate contract breaches, based on predictions that extrapolate current measurements, and trigger corrective actions, aiming at avoiding the breach altogether, or at least at mitigating its effects. In this paper we propose a novel QoS prediction approach which combines run-time monitoring of the real system with probabilistic model-checking on a system model. To achieve accurate predictions while limiting the state explosion problem (which is an intrinsic limitation of model-checking techniques), we build a probabilistic model of the business process based on an analysis of the workflow on which it relies. System parameter values are thus extracted from measurements that are collected on the field. Combined use of direct measurements and analytical modelling has proven to be an effective system analysis approach. The model checker then uses the actualized system model to estimate the probability that in the near future (i.e. in the range of a few minutes) the system reaches a status that would result in an SLA violation. Since i) the amount of data that is collected by the monitoring infrastructure that observes the real system can reach high volumes very quickly [30], and ii) parameter values must be extracted in a timely fashion (otherwise they would be useless), we rely on big data analytics solutions. To validate the proposed approach, a prototype of the QoS prediction framework has been developed, and an experimental campaign has been conducted with respect to a case study in the field of Smart Grids [10, 8].

The paper is organized as follows. Section 2 describes related work on QoS prediction. In section 3 we present the overall architecture enabling our QoS prediction approach, whereas we show the internal design of the runtime quality prediction prototype we developed to validate our approach in section 4. Section 5 illustrates the case study to which we applied the proposed methodology. Finally, section 6 closes the paper with conclusions and future work.

2 Related Work

QoS prediction is surveyed in [24], [17], [22], and [23]. A prediction performance model is treated in [24], where the authors exploit the Markovian Arrival Process (MAP) and a MAP/MAP/1 queuing model as a means to predict performance of servers deployed in Cloud infrastructure. Although in our Smart Grid case study we use a M/M/1 queuing model, our QoS prediction methodology does not rely on a specific model which, therefore, could be adapted as needed. A prediction-based resource measurement which use Neural Networks and Linear Regression as techniques to forecast future resource demands is proposed in [17]. A regression model is used also in [23] to produces numerical estimation of Service Level Objectives (SLOs) so as to predict SLA violations at runtime. Similarly, the PREvent framework, a system which uses a regression classifier to predict violation, is presented in [22] but details about the performance of the method are not given. In [4, 26, 28] QoS requirements are controlled by solving a QoS optimization problem at runtime. Particularly, in [4] a linear programming optimization problem is adopted to define a runtime adaptation methodology for meeting QoS requirements of service-oriented systems, whereas a multi-objetive optimization problem to develop QoS adaptive service-based systems for guaranteeing pre-defined QoS attributes is proposed in [26] and [28] .

A collaborative method is proposed in [29] where performance of cloud components is predicted based on usage experiences. Although this method could be appropriate for QoS indicators from the user perspective, it is impractical in the general case where QoS are business-oriented.

A QoS prediction by using a Model Checking solution is proposed in [14], and [15]. [14] proposes an approach named ATOP - i.e. from Activity diagrams TO Prism models - which from an abstract de-

scription of service compositions (activity diagram) derives a probabilistic model to feed the PRISM tool for the evaluation phase. However, unlike our solution this is a methodology conceived for evaluating the system at design-time. Similarly to our work, [15] proposes a two-phases method involving monitoring and prediction with the aim of monitoring at run-time the reliability of compositional web services which exhibits random behaviour. Although this method also takes advantage of the probabilistic model checking technique, it mainly focuses on reliability estimation by using a DTMC-based Markovian model. In contrast, we propose a general CTMC probabilistic model for performance indicators in which both states and transitions are parametrized, resulting in a model which can be adapted at run-time.

3 Architectural Overview

This section describes the approach behind our solution for QoS prediction. An high level architecture overview is represented in fig. 1.

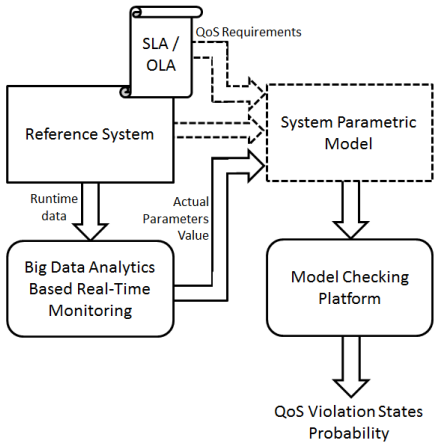


Figure 1: The QoS Monitoring and Prediction achitecture

Given a system/process to be monitored for QoS compliance with a set of SLAs and OLAs, we assume that a formalized model of the system is made availabe. Such a model is based on a state-transition description which is able to capture the evolution of Key Performance Indicators (KPIs) over time. Moreover states and transitions must be expressed as parameters. The KPIs can be inferred by the SLAs and OLAs defining the expected QoS. It can thus be used to identify the conditions of violation of the expected QoS. Such conditions are represented by some final states in the state-transition model.

At run-time, the reference system is continously monitored and collected data are used to evaluate the actual value of model parameters. Once the model has been populated with estimated values of the parameters, it is processed by the model checking software. In our prototype we have used PRISM [21] which is an open-source probabilistic model checker which supports the analysis and checking of a wide number of model types. The model checker explores states that can be reached since current state in a fixed number of transitions (depending on the the desired prediction time-lapse). If one of the states representing a violation is likely to be reached with a probability higher than a fixed threshold (violation alarm threshold), than a QoS breach is predicted.

It is worth noting that the usage of a parametric model, which is continuously updated, and the fixed time-lapse used for the prediction, allow to limit the well-known state explosion problem due to the exhaustive states exploration operated by model-checkers. One further optimization could be operated

by pruning those branches including states reachable with a probability lower than the violation alarm threshold.

As for the parameters evaluation, continous monitoring of a complex system may require the real-time analysis of huge amounts of data. Such requirement can be matched by using advanced Big Data Techniques and tools. In particular, in our prototype we used a Complex Event Processor (CEP) to infer parameters value from collected data. In an advance prototype the Big Data layer could be used to support the model-checking process.

To guarantee that the automatic procedure be both efficient and consistent, two conditions are to be held:

- the size of the state space of the model has to be sufficient to perform the model-checking analysis in a time that is compatible with the updating time of the QoS data of the modelled system
- the evaluation of the model parameters should always allow the representation of the critical QoS states to be monitored.

The first condition is key for obtaining a near real-time QoS prediction system. Indeed, it requests to balance the size of the QoS model at run-time by taking into account both the real time constraint imposed by the monitored service and the time spent to model check. A preliminary analysis during the model definition has to be conduct in order to ensure that this condition is still true even though the model is fully expanded (i.e. no pruning of its state space is considered). The second condition allows to verify that, if narrowed, the model still includes states of the real system related to critical QoS values (e.g. warning and/or violation states).

Consequently, our methodology considers the following steps:

1. Specification of the parameterised QoS stochastic model and QoS constraints to monitor
2. Real-time data analysis and parameters synthesis
3. Generation of the internal state-transition Model representation
4. Execution of the Probabilistic Model-Checking to quantify the likelihood of future QoS state
5. QoS Verification

In the first step we define a stochastic model which is suited to the kind of properties we are interested in monitoring. In this paper we show a case study, from the Smart-Grid domain, modelled by means of a CTMC. The steps 2-5 are involved in an endless loop which makes our approach adaptive. In particular, the second step needs to analyse data received by the CEP so to determine the parameters of the model, and computes the current KPIs value. The third step generates the finite state-transition representation of the system model on which performing model-checking in the fourth step. Finally, the fifth step deals with verifying the QoS on the basis of the current KPIs and/or quantification of future QoS states.

3.1 QoS Properties Specification

In a previous work we introduced the concept of Quality Constraint (QC) [7] as a mean to express constraints on KPIs. A QC is defined as a boolean condition on a single KPI. The language we used to specify QCs is an interval-based version of the Linear-time Temporal Logic (LTL). Particularly, in [7] we introduced two temporal operators *along* and *within* which present the following semantic:

- P **along** T: P is true in any time instant belonging to T

- P **within** T: there exists at least a time instant $i \in T$ in which P is true

Thus, the **along** and **within** are, respectively, the restriction of the Linear Temporal Logic (LTL) "globally" (G) and "eventually" (F) operators to the interval T. A QC without temporal operator is interpreted as an expression to be verified all along the lifetime of the monitored system, hence resulting useful for specifying safety property.

It is worth noting that in the context of runtime monitoring we check properties against execution traces of the system, i.e ordered sequences of past (up to now) states. Although in this way we are able to recognise a violation as soon as it happens, we do not have any means to evaluate if it will happen and when in the future.

In this model-based approach we tackle this issue by defining Predictive Indicators (PIs) upon the monitored KPIs. A PI is a numerical indicator which statistically quantify the probability for a KPI to be in a certain state (i.e. a range of values) in a predetermined time instant in the future. Taking advantage of probabilistic model-checking we define such PIs as probabilistic temporal formulae (in the logic suitable for the underlying model) which can be evaluated over all possible evolution considered in the service model. Furthermore, as numerical indicators PIs can be monitored by means of specifying QCs. To this purpose we have extended our QC language with the **eval(ϕ)** operator which accept temporal logic formula ϕ to be evaluated by means of a model checker tool. As a predictive quality indicator, **eval(ϕ)** can be monitored by specifying a Quality Constraint as we will see in the Smart Grid case study.

3.2 Performance Model

In this work we focus our attention on KPIs which refers to quantifiable service performances, i.e. resource utilization, number of served requests, etc. To better fit our case study, we selected a M/M/1 queuing model to represent these type of indicators. The intuition is that such indicators represent resources whose arrival usage requests are determined by a Poisson process of parameter λ, whereas the resource service time follows an exponential distribution of parameter μ.

Let us assume a KPI as a variable k whose values can range in set V_K seen as:

$$k \in V_K = A_V \cup C_V \cup I_V$$

where the subsets A_V, C_V and I_V have the following meaning:

A_V**:** it is the set of *Admissible Values* k takes when the system is in a state which fulfills all the QCs defined on the KPI

C_V**:** it is the set of *Critical Values*, i.e. limits/targets values, on which the system still meets the required quality but beyond which this is no longer true.

I_V**:** it is the set of *Inadmissible Values* k takes when the system is in a state which does not fulfill at least a QCs defined on the KPI

We assume that V_K is totally ordered and its subsets are disjoint, that is:

$$\forall a,b,c : a \in A_V, b \in C_V, c \in I_V \text{ s.t. } a < b < c$$

The fig.2 illustrates our general queueing model. We consider a queue as a discrete representation of the set V_K. In particular, the V_K is partitioned into a sequence of N disjoint intervals $I_i = [a_i, b_i]$, $i \in 0,..,N-1$ with $|I_i| = b_i - a_i = \frac{|V_K|}{N}$. Moreover, for $b_i \in I_i$, $a_{i+1} \in I_{i+1}$ for all $i \in 0,..,N-2$, we have $b_i < a_{i+1}$. This helps to preserve the semantic distinction among the subsets A_V, C_V and I_V. Hence, we can write $I_i < I_j$ if $i < j$.

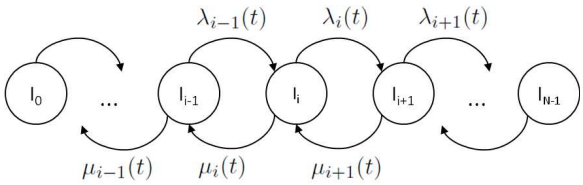


Figure 2: The general form of the adopted queueing model

Thus, let t be the total amount of elapsed time from the beginning of KPI monitoring, $w = t - T$ the time window, with $T < t$, in which we take into account the KPI variations, and k_{t_1}, k_{t_2}, $t_1 < t_2$ two sequential values of the KPI of interest belonging respectively to the interval I_i and I_j with $i < j$. We adapt the queueing model by interpreting:

- the queue length $L_Q = i$ as representing the interval I_i in which k_{t_1} lies
- given all the transitions I_i to I_j with $i < j$ (resp. $i > j$) observed up to the time instant t, the increment (resp. decrement) rate λ_t (resp. μ_t) is $\sum_{i<j} \frac{j-i}{w}$ the ratio of the sum of all increments $j-i$ (resp. decrement $i-j$) over the time window we want to consider for the rating updating.

Therefore, the queue length increases from $L_Q = i$ to $L_Q = i+1$ for $i = 0,..,N-1$ with a rate λ_t and decreases from $L_Q = i$ to $L_Q = i-1$ for $i = 1,..,N$ with a rate μ_t. An M/M/1 queue model can be described by an CTMC. In this way, by using the Continuous Stochastic Logic (CSL) as a language to formally specifying properties, we employ the probabilistic model-checking technique to conduct a quantitative analysis on the KPIs by means of their queue representation.

4 QoS Prediction Internals

To validate our methodology, a Runtime Quality Prediction (RQP) prototype has been developed. In this section we detail its internal design. Fig. 3 shows architectural interactions between monitoring and prediction systems, as well as interactions among their components. For the sake of space, the figure only shows the components of the monitoring system interacting with the RQP.

The RQP tool relies on the following modules: RQP-manager, Parameters Meter, Modeller, and CTMC Model Checker. In the following we describe their main activities.

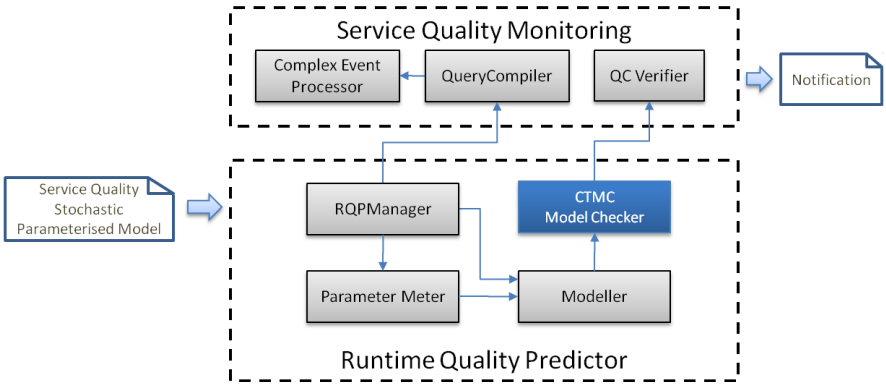


Figure 3: Monitoring-Prediction Architecture

- **RQP-Manager**: it is responsible for all the activities carried out by the predictor. It manages the pre-evaluation phase by parsing the description model, calling the QueryCompiler function `State2Query()` to generate the query related to the states of the model, and executing a simulation run of the system in order to measure its execution time.
- **Parameters Meter**: it is in charge of computing the model parameters based on the fresh data continuously collected by the monitoring system. Such component works as a controller making it possible to tune the frequency by which it sends the updated value to the Modeller. Moreover, it also allows to define a metric based on the parameters variation so to avoid to perform model checking if, for instance, the current state is not changed and changes to the transition rate matrix are below a certain predefined threshold.
- **Modeller**: its main task is to replace the new parameters value in the abstract representation of the model, and then to generate the state space with respect to the model representation used by the underlying model checker (e.g. explicit, symbolic, hybrid).
- **CTMC Model Checker**: it performs the CTMC model checking algorithm with the updated model and the reachability properties given as input.

The tasks executed in the pre-evaluation phase are the following:

1. the description model and the reachability properties received as input are parsed. This includes analyzing syntactically both model and properties, and building an abstract internal representation in which, starting form the monitored data, the information about the state-transition model, their parameters, and the rules useful to identify the initial system states are saved.
2. the state-identification rules are translated into CEP-based queries and submitted to the CEP.
3. a simulation run is performed in order to evaluate the execution time of the run-time phase
4. if the simulation step shows satisfiable performance, then the quality constraint of the form $t_e < T$ - defined on the execution time indicator t_e of the runtime prediction - is generated and registered into the QC Verifier.

Once the pre-evaluation phase has been executed and the CTMC-based model accepted, the prediction can be activated. Fig. 4 depicts the monitoring-prediction process executed during the run-time phase. The steps performed during this phase are the following:

1. the CEP collects and pre-processes events delivered by the monitored service. Particularly, it performs the queries which identify the current quality state of the service, and sends such information to the Parameter Meter. The current observed state is also delivered to the Modeller in that it will consider it as the actual initial state.
2. based on the inter-arrival time and on the transition occurred, the Parameter Meter updates the transition rate matrix. The parameters variation is evaluated against a predefined threshold in order to avoid the regeneration of the model in case that minimal changes occur.
3. the Modeller receives the currently observed state and the updated transition rate matrix, and updates the model parameters. Then, it regenerates the whole state space.
4. the CTMC model checking is performed against the updated model, and the reachability properties are probabilistically quantified, i.e. predictive indicators are assessed.

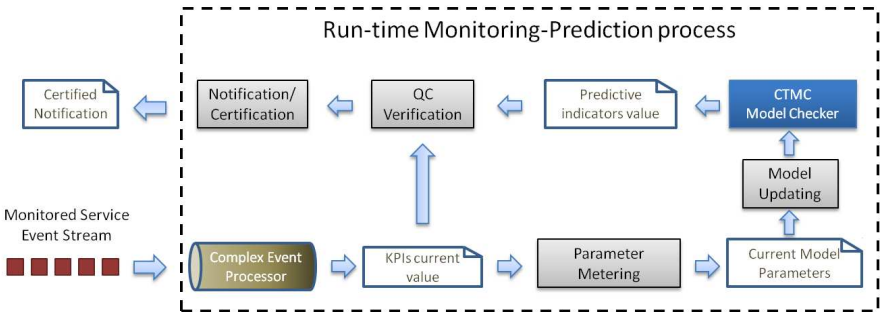


Figure 4: The Quality Monitoring and Prediction process

5. the QC Verifier receives the updated values of predictive indicators and checks them against the associated quality constraints.

The RQP prototype has been developed as an autonomous application with respect to the monitoring system. In the following we give a description of its internal design.

The RQP application employs the CTMC model checking algorithm realised by the probabilistic model checker PRISM. In comparison with other model checkers which support probabilistic Markovian models such as MRMC[18], APMC[16] , PAT [25], the choice of PRISM has been dictated by three important aspects that characterise this tools. Firstly, it implements various model checking algorithms according to the internal representation of the state-transition model, i.e. explicit, symbolic, hybrid, allowing to have a great flexibility with regards to model size. Furthermore, it allows to choose different iterative numerical resolutions (e.g. Jacobi, Gauss-Seidel), and to specify the desired precision by setting a specific parameter. Secondly, besides supporting a command-line interaction, it is equipped with a rich graphical user interface (GUI). Such GUI provides a complete visual environment wherein to specify the model and to perform qualitative/quantitative analysis as well as simulations. This gives an important support for designing and evaluating the system model in advance. Furthermore, PRISM is a well designed tool which provides a clear Java-based API layer by which taking advantage of all its functionalities. Finally, it is a widespread tool well documented, supported, and broadly used in many areas of science and engineering[20].

Before examining in detail the internal design of the RQP, let us give a quick overview of the PRISM model checker in order to grasp the main concepts. For more details see the References section.

4.1 PRISM Model Checker

PRISM [19, 21] is an open-source probabilistic model checker which supports the analysis and checking of a wide number of model types: discrete and continuous time Markov chain (DTMC and CTMC respectively), Markov decision process (MDP), probabilistic timed automata (PTA). The models are specified in a textual modelling language based on reactive modules formalism[1]. It allows to: specify: (i) the set of modules representing different system components; (ii) the probabilistic behaviour of each module by a state-transition description; (iii) the synchronous/asynchronous composition among modules. The properties to check are expressed by using the appropriate language according to the underlying model. For the purpose of our prediction methodology, we have introduced only reachability property with bounded temporal operators expressed in PRISM by the probabilistic operator "$\mathscr{P}$" and the temporal one "F". For example, to specify the quantitative reachability property "the probability that a state `available` is reached within 15 units time", we write `P=?[F<15 (available)]`. The question mark induces the model checking algorithm to execute the reachability analysis with the aim of quantify the probability to satisfy the formula. On the contrary, for a qualitative reachability property we can

specify the desired likelihood as in "the probability that a state `available` is reached within 15 units time is at least 0.95", written as `P>=0.95[F<15 (available)]`.

The tool can handle very large models (10^{100} states) because it can use symbolic model checking techniques. For this purpose, it employs binary decision diagram (BDD) data structure for internally representing the state-transition model. In particular, PRISM uses a generalisation of BDDs called multi-terminal BDD (MTBDDs) [2].

Internally PRISM works as follows. First, it parses the model description and the provided properties, and builds an abstract representation of the state-transition model which is independent of the particular model checking technique selected. Then, from such abstract data structure, it is generated the specific representation - explicit, symbolic, or hybrid - against which to check the properties. Finally, after the execution of the appropriate model checking algorithm with regards to the internal representation, PRISM reports a true/false outcome for qualitative properties, which indicates whether or not the system model satisfies the given property. As for quantitative properties, PRISM returns numeric results which quantify the likelihood for the property to be satisfied (see e.g. [19]).

4.2 Runtime Quality Predictor prototype

Fig 5 shows the class diagram of our RQP along with the main PRISM classes which our prototype interacts with. PRISM implements about a hundred classes. Here are pictured only those strictly related to our prototype and to the purpose of this dissertation. The `Prism` class realises the high level application functionalities and links all the main classes which declare/define the major procedures such as parsing, model generation, model checking, and so forth. Two important classes are `Model` and `ModelChecker` (both defined as Interfaces) which describe the behavioural features associated with, respectively, an abstract model and model checking technique. The two concrete classes interesting for our prototype are `CTMC` and `StochModelChecker`. The former contains data and methods to handle CTMC model, whereas the latter implements the probabilistic model checking functions.

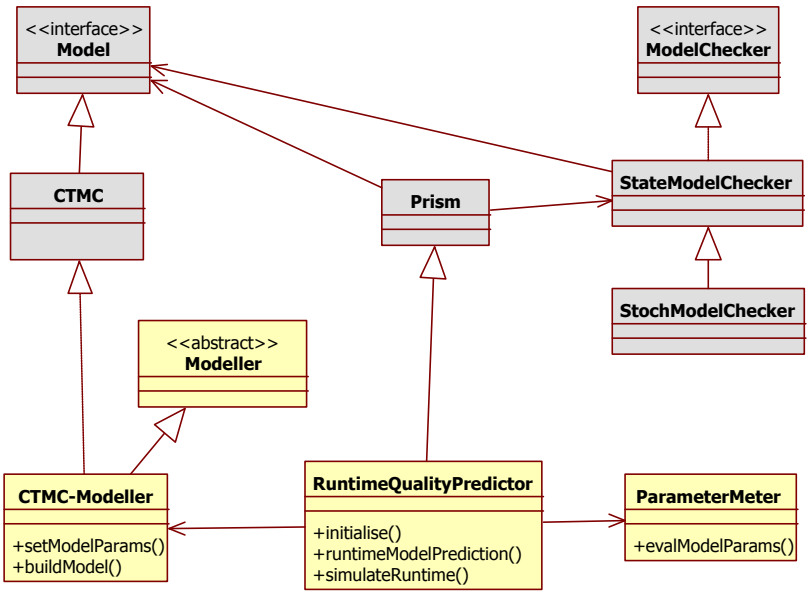


Figure 5: Runtime Quality Prediction: Class Diagram

`RuntimeQualityPredictor` is the main class in our prototype. It inherits from the `Prism` class representing and managing the whole application as it runs. In fig. 5 we report the main methods

implemented in each class. In detail, the `initialise()` method deals with setting up the application as well as the underlying PRISM. `simulateRuntime()` realises the pre-evaluation phase of our prediction methodology. The current implementation does not include a complete process simulation, i.e. the emulation of service events and CEP processing is not realised. In the actual implementation such method evaluates the execution time of the following three steps: parameter evaluation (method `evalModelParams()` of class `ParamterMeter`), model building (method `setModelParams()` of class `CTMC-Modeller`), and model checking (method `modelCheckProperties()` of PRISM class `StateModelChecker`).

The `runtimeModelPrediction()` is the principal method which implements the runtime phase of the RQP. At a very high level the algorithm is straightforward (algorithm 1).

Algorithm 1 Runtime Quality Prediction

```
1:  function RUNTIMEMODELPREDICTION
2:      /* Initialisation */
3:      ...
4:      while true do
5:          res ← RECEIVEQUERIESRESULT()
6:          currentState ← EVALCURRENTSTATE(res)
7:          [changed, pars] ← EVALMODELPARAMS(res,eps)
8:          if changed OR (currentState<>oldState) then
9:              SETMODELPARAMS(currentState,pars)
10:             BUILDMODEL()
11:         end if
12:         resProps ← MODELCHECKPROPERTIES(props)
13:         breach ← QCVERIFIER(resProps)
14:         if (breach) then
15:             /* auxiliary utilities */
16:             break
17:         end if
18:         oldState ← currentState
19:     end while
20: end function
```

The loop (lines 4-19) iterates endlessly and is stopped if the breaking condition (lines 14-17) becomes true. In line 5 RQP receives the outputs of the queries processed by the CEP. We recall that the interaction between RQP and CEP is event-based. The thread which executes `runtimeModelPrediction()` works synchronously with respect to the CEP queries. As soon as the results of the query processing are received by the RQP, they are passed to the `ParameterMeter` object which, in turn, computes the current state and the new values of the transition rate matrix (lines 6 and 7). The parameter `eps` indicates the minimum variation to consider for regenerating the model. Internally `evalModelParams` evaluates this variation and returns a boolean result encoded by the `changed` variable. If either the current state or the transition rate matrix is changed (line 8), the model parameters are updated (`setModelParams` function at line 9) and the state space is generated (`buildModel` function at line 10). At this stage, the model checking phase can be executed on the quantitative reachability properties defined by the user (line 12). The results of this phase are sent to the QC Verifier component (line 12) of the monitoring system in order to check the predictive indicators against predefined constraints. In the current implementation the auxiliary utility (line 14) deals with saving the RQP state for debugging reason.

5 The Smart Grid Case Study

The proposed QoS Prediction approach has been validated with respect to a Smart Grid (SG) case study.

SG is the integration of the IT infrastructure into a traditional power grid in order to continuously exchange and process information to better control the production, consumption and distribution of electricity. For this purpose Smart Meters (SMs) devices are used to measure variations of electric parameters (e.g. voltage, power, etc.) and send such data to a computational environment which, in turn, analyse and monitor it in a real-time fashion.

In this case study, our tool performs the remote monitoring on behalf of an Energy Distributor (ED) which purchases electric power from Energy Producers (EPs) and retails it to Energy Consumers (ECs). The primary goal of the ED is to balance the purchased electric power with respect to the variations of power demand.

The SG Model. For the sake of simplicity we have built a basic model which represents an ED, EP (or aggregated values of many EPs) and EC (or aggregated values of many ECs) as a three-queue system networked as in fig. 6. Each queue is a discrete representation of the real-valued KPI to be modelled.

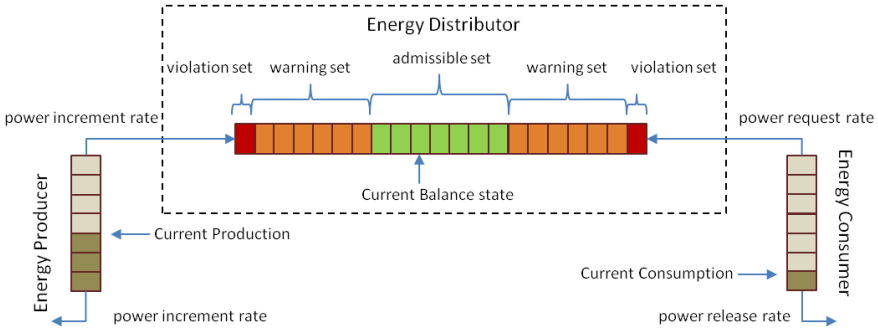


Figure 6: Network queuing system model

The PRISM-based model we define implements the queues ED_Q, EP_Q, and EC_Q with queue length and transition rates as parameters. Furthermore, the sets A_V, C_V, I_V are arranged as follows:

$$\begin{aligned} A_V &= \{d_e : d_e \geq adm_{min} \text{ and } d_e \leq adm_{max}\} \\ C_V &= \{d_e : -cri_{min} \leq d_e < adm_{min}\} \cup \\ &\quad \cup \{d_e : adm_{max} < d_e \leq cri_{max}\} \\ I_V &= \{d_e : d_e < -cri_{min} \text{ or } d_e > cri_{max}\} \end{aligned}$$

where adm_{max}, adm_{min} and cri_{min} and cri_{max} represent the minimum and maximum thresholds of the admissible and critical value sets.

Parameters Updating. In the queuing model the queue length and the transition rate are updated as follows. Two thresholds are set on both queue edges so that if the current state goes below the first or up the second, the queue length is doubled or halved respectively. As for the updating of the transition rate, an Exponential Weighted Moving Average (EWMA) is applied on the first difference of the time series under analysis. Thus, let $Y = y_1, y_2, \ldots$ a time series, we compute the transition rate ρ as follows:

$$\rho' = \alpha(y_i - y_{i-1}) + (1-\alpha)\rho \tag{1}$$

in which the initial value of ρ is set to 0. The 1 is used for both the increment (μ_t) and the decrement rate (λ_t).

QoS Data Extraction. To tackle the Big Data issue, our architecture takes advantage of a Complex Event Processing (CEP) which could be performed on any data-intensive distributed framework (e.g. Hadoop). Such combination allows to extract, process and deliver (complex) data in real-time, empowering the QoS monitoring and prediction phases. Following our case study, we show an example of a complex event to derive the *balance indicator* (BI) from the basic SmartMeterMeasureEvent originating from the Smart Meters of EPs and ECs:

```
insert into BalanceIndicatorEvent
select (EP.measure - EC.measure) as index
from EP.SmartMeterMeasureEvent as EP,
 EC.SmartMeterMeasureEvent as EC,

select "range_i"
from BalanceIndicatorEvent
where index index >= I_min and <= I_max
```

The first query compute the BI and create the event `BalanceIndicatorEvent`. The second query is a template used by our QoS Monitoring tool to generate the actual queries. They are used to classify at which range the index belongs to. Based on this data, the QoS Analyser component compute the transition rate from one range to another to be fed the PRISM model. On the other side, a temporal-based query is used for a real-time anomalous detection:

```
select measure, "CriticalValueMsg"
from EP1.SmartMeterEvent.win:time(15 min)
where measure < BASE_PROD
```

In this case we take advantage of the temporal-based capability of the CEP language. The select deliver a `CriticalValueMsg` message based on the fact that a specific energy producer (EP1 in the example) is gone underproduction. The message is delivered to the QoS Monitoring which in turn perform the associated action, e.g. notify ED.

Quality Constraints to be Monitored. Briefly we report only two types of QCs: the first is a safety property (neither *within* nor *along* operator specified) which assesses if the predicted violation probability in the next 15 minutes is more than 10%.

$$\textbf{eval}(\mathscr{P}_{\geq 0.1}[F_{\leq 30}\texttt{"violState"}]) = \texttt{false} \tag{2}$$

The second QC guarantees to be notified if the probability of incurring in a violation state in the next 30 minutes is greater than 0.05 twice in a row (considering a measurement events every 15 minutes).

$$\textbf{eval}(\mathscr{P}_{=?}[F_{\leq 30}\texttt{"violState"}]) \leq 0.05 \textbf{ within } 30m \tag{3}$$

It is worth noting that the **within** operator allows to express Quality Constraints on predicted values at the run-time level, which are computed by different executions of model checking.

5.1 Validation

In our scenario we assume a balance range of 800 Mega Watt (MW), i.e. $[min_b = -400, max_b = 400]$, and we firstly evaluate how much time the QoS prediction phase takes with respect to different model size (Table 1). The table also reports the size of the model in terms of number of states and transitions. As expected by using a model-ckecking technique, the time is exponential against the model size. However, as the last row shows, we can also observe that even in case of millions (10^6) of states and transitions

Queue length	#States	#Trans	BM time (s)	MC time (s)	Tot. time (s)
10	2040	9336	0.03	0.08	0.11
20	13280	63476	0.15	0.38	0.53
30	41720	202416	0.35	2.44	2.79
40	95360	466156	0.81	8.37	9.18
50	182300	894696	3.17	27.01	30.18
60	310240	1528036	7.75	52.51	60.27
70	487480	2406176	13.39	135.49	148.88
80	721920	3569116	20.53	197.11	217.64
90	1021560	5056856	27.53	355.29	382.82
100	1394400	6909396	42.80	492.28	535.089

Table 1: Queues length, model size, and execution time

- that means a fine-grained discretisation - the total time is less than 9 minutes, hence still comparable with the updating rate usually considered for SGs.

We have selected a queue length of 40 - i.e. a unit increment/decrement of the queue correspond to a 20MW of balance variation - and set these thresholds: $adm_{min} = -200$, $adm_{max} = 200$, $cri_{min} = -380$, $cri_{max} = 380$. Our tests are based on property 3 evaluated by simulating three different scenarios:

Case A: EPs inject in the grid as much energy as ECs need (balanced case).

Case B: ECs request less than EPs produce (overproduction).

Case C: The energy consumption request rises twice compared with the production rate (imbalanced condition).

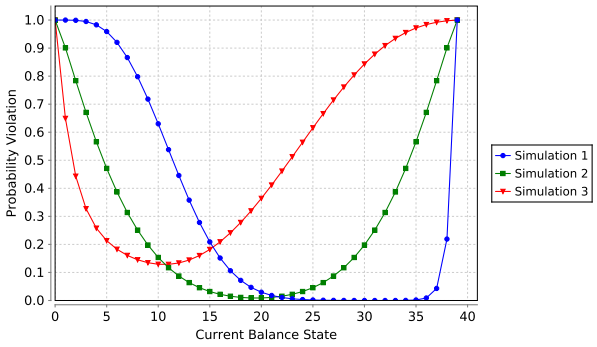


Figure 7: Violation Probability (queue length=40)

Fig. 7 plots the violation probability estimated for such scenarios. For scenario A the violation probability varies in a symmetrical fashion around the balance point (i.e. queue length 20). The scenario B exhibits a higher probability in all the overproduction states (i.e. queue length less than the balance point), and a lower one for a large number of states representing the power grid overload (i.e. queue length greater than the balance point). This characteristic is emphasised in the third simulation which represents the imbalanced (overloaded in this case) scenario. In addition, we notice how in such anomalous conditions all minimum values of violation probability are higher than the other two scenarios.

6 Conclusion and Future Work

To support Big Data analysis of QoS information, in this paper, we have proposed a QoS prediction framework which takes advantage of the qualitative and quantitative analysis performed by a probabilistic model-checking technique. Our approach uses a parametric QoS model and performs a probabilistic model-checking analysis in order to evaluate QoS-related predictive indicators (PIs). In this way, pre-alert QoS states can be notified in advance, giving a greater control to the Service Provider to avoid, or at least manage, possible breaches of Service Level Agreements (SLAs) contracted with Service Consumers. We have realized and presented a validating prototype - built on top of the PRISM Model Checker -, as well as experiments on a Smart Grid case study, which shows the effectiveness of our methodology and how, tuning the model parameters, the time required to model check is less than the time needed to receive updated QoS information from the monitored service. In the next future we plan to extend the experimental campaign validating our approach and to extend the usage of this framework to monitor security [9] and other non-functional aspects other than provided QoS.

Acknowledgments

This work has been partially supported by the TENACE PRIN Project (n. 20103P34XC) and by the LIMS PON project (PON03PE_00159_5) funded by the Italian Ministry of Education, University and Research

References

[1] R. Alur and T. A. Henzinger. Reactive modules. *Formal Methods in System Design*, 15(1):7–48, July 1999.

[2] R. Bahar, E. Frohm, C. Gaona, G. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebric decision diagrams and their applications. *Formal Methods in System Design*, 10(2-3):171–206, April 1997.

[3] F. Campanile, L. Coppolino, S. Giordano, and L. Romano. A business process monitor for a mobile phone recharging system. *Journal of Systems Architecture - Embedded Systems Design*, 54(9):843–848, September 2008.

[4] V. Cardellini, E. Casalicchio, V. Grassi, F. Lo Presti, and R. Mirandola. Qos-driven runtime adaptation of service oriented architectures. In *Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering (ESEC/FSE '09), Amsterdam, The Netherlands*, pages 131–140. ACM, August 2009.

[5] A. Ceccarelli, M. Vieira, and A. Bondavalli. A testing service for lifelong validation of dynamic soa. In *Proc. of the 13th IEEE International Symposium on High-Assurance Systems Engineering (HASE)*, pages 1–8. IEEE, Nov 2011.

[6] G. Cicotti, L. Coppolino, R. Cristaldi, S. D'Antonio, and L. Romano. Qos monitoring in a cloud services environment: The srt-15 approach. In *Proc. of the Parallel Processing Workshops (Euro-Par), Bordeaux, France*, volume 7155, pages 15–24. Springer Berlin Heidelberg, August-September 2012.

[7] G. Cicotti, S. D'Antonio, R. Cristaldi, and A. Sergio. How to monitor qos in cloud infrastructures: The qosmonaas approach. In *Proc. of the the 6th International Symposium on Intelligent Distributed Computing (IDC 2012), Calabria, Italy*, volume 446, pages 253–262. Springer Berlin Heidelberg, September 2013.

[8] L. Coppolino, S. D'Antonio, I. A. Elia, and L. Romano. Security analysis of smart grid data collection technologies. In *Proc. of the 30th international conference on Computer safety, reliability, and security (SAFECOMP'11), Naples, Italy*, pages 143–156. Springer-Verlag, Berlin, Heidelberg,, September 2011.

[9] L. Coppolino, S. D'Antonio, V. Formicola, and L. Romano. Integration of a system for critical infrastructure protection with the ossim siem platform: A dam case study. In *Proc. of the 30th International Conference on Computer Safety, Reliability, and Security (SAFECOMP 2011), Naples, Italy*, volume 6894, pages 199–212. Springer Berlin Heidelberg, September 2011.

[10] L. Coppolino, S. D'Antonio, and L. Romano. Exposing vulnerabilities in electric power grids: An experimental approach. *International Journal of Critical Infrastructure Protection*, 7(1):51—60, March 2014.

[11] L. Coppolino, S. D'Antonio, L. Romano, F. Aisopos, and K. Tserpes. Effective qos monitoring in large scale social networks. In *Proc. of the 7th International Symposium on Intelligent Distributed Computing (IDC 2013), Prague, Czech Republic*, volume 511, pages 249–259. Springer International Publishing, September 2014.

[12] L. Coppolino, D. De Mari, L. Romano, and V. Vianello. Sla compliance monitoring through semantic processing. In *Proc. of the 11th IEEE/ACM International Conference on Grid Computing (GRID)*, pages 252–258. IEEE, Oct 2010.

[13] L. Coppolino, L. Romano, N. Mazzocca, and S. Salvi. Web services workflow reliability estimation through reliability patterns. In *Proc. of the 3rd IEEE International Conference on Security and Privacy in Communications Networks and the Workshops, (SecureComm)*, pages 107–115. IEEE, Sept 2007.

[14] S. Gallotti, C. Ghezzi, R. Mirandola, and G. Tamburrelli. Quality prediction of service compositions through probabilistic model checking. In *Proc. of the 4th International Conference on the Quality of Software-Architectures (QoSA), Karlsruhe, Germany,*, volume 5281, pages 119–134. Springer Berlin Heidelberg, October 2008.

[15] H. Gao, H. Miao, and H. Zeng. Predictive web service monitoring using probabilistic model checking. *International Journal on Applied Mathematics & Information Sciences*, 6(1L):139–148, Feb. 2013.

[16] T. Hérault, R. Lassaigne, F. Magniette, and P. S. Approximate probabilistic model checking. In *Proc. of 5th International Conference In Verification, Model Checking, and Abstract Interpretation (VMCAI 2004) Venice, Italy*, volume 2937, pages 73–84. Springer Berlin Heidelberg, January 2004.

[17] S. Islam, J. Keung, K. Lee, and A. Liu. Empirical prediction models for adaptive resource provisioning in the cloud. *Future Generation Computer Systems*, 28(1):155–162, Jan 2012.

[18] J. P. Katoen, M. Khattri, and I. S. Zapreevt. A markov reward model checker. In *Proc. of the 2nd International Conference on the Quantitative Evaluation of Systems (QUEST05), Turin, Italy*, pages 243–244. IEEE Computer Society, Sept. 2005.

[19] M. Kattenbelt, M. Kwiatkowska, G. Norman, and D. Parker. Abstraction refinement for probabilistic software. In *Proc. of the 10th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI'09)*, volume 5403, pages 182–197. Springer Berlin Heidelberg, January 2009.

[20] M. Kwiatkowska, G. Norman, and D. Parker. Stochastic model checking. In *Proc of the 7th International School on Formal Methods for the Design of Computer, Communication, and Software Systems (SFM 2007), Bertinoro, Italy*, volume 4486, pages 220–270. Springer Berlin Heidelberg, May-June 2007.

[21] M. Kwiatkowska, G. Norman, and D. Parker. Prism 4.0: Verification of probabilistic real-time systems. In *Proc. of the 23th International Conference on Computer Aided Verification (CAV), Snowbird, UT, USA*, volume 6806, pages 585–591. Springer Berlin Heidelberg, July 2011.

[22] P. Leitner, A. Michlmayr, F. Rosenberg, and S. Dustdar. Monitoring, prediction and prevention of sla violations in composite services. In *Proc. of the IEEE International Conference on Web Services (ICWS)*, pages 369–376. IEEE, July 2010.

[23] P. Leitner, B. Wetzstein, F. Rosenberg, A. Michlmayr, S. Dustdar, and F. Leymann. Runtime prediction of service level agreement violations for composite services. In *Proc. of the 2009 International Conference on Service-oriented Computing (ICSOC/ServiceWave'09), Stockholm, Sweden*, pages 176–186. Springer-Verlag, November 2009.

[24] S. Pacheco-Sanchez, G. Casale, B. Scotney, S. McClean, G. Parr, and S. Dawson. Markovian workload characterization for qos prediction in the cloud. In *Proc. of the IEEE International Conference on Cloud Computing (CLOUD)*, pages 147–154. IEEE, July 2011.

[25] J. Sun, Y. Liu, J. Dong, and J. Pang. Pat: Towards flexible verification under fairness. In *Proc. of the 21st International Conference on Computer Aided Verification (CAV 2009), Grenoble, France*, volume 5643, pages 709–714. Springer Berlin Heidelberg, June-July 2009.

[26] C. Tao, B. Rami, and T. Georgios. Dynamic qos optimization architecture for cloud-based dddas. In *Proc. of the International Conference on Computational Science*, volume 18, pages 1881 – 1890. Elsevier B. V., 2013.

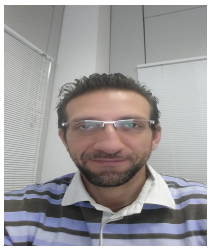
[27] W. Tsai. Service-oriented system engineering: a new paradigm. In *Proc. of the 1st IEEE International Workshop on Service-Oriented System Engineering (SOSE)*, pages 3–6. IEEE, Oct 2005.

[28] S. Yau, N. Ye, H. Sarjoughian, and D. Huang. Developing service-based software systems with qos monitoring and adaptation. In *Proc of the 12th IEEE International Workshop on Future Trends of Distributed Computing Systems (FTDCS)*, pages 74–80. IEEE, Oct 2008.

[29] Y. Zhang, Z. Zheng, and M. Lyu. Real-time performance prediction for cloud components. In *Proc. of the 15th IEEE International Symposium onObject/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW)*, pages 106–111. IEEE, April 2012.

[30] Z. Zibin, Z. Jieming, and M. Lyu. Service-generated big data and big data-as-a-service: An overview. In *Proc. of the 1st IEEE International Congress on Big Data (BigData Congress), Santa Clara Marriott, CA, USA*, pages 403–410. IEEE, June 2013.

Author Biography



Giuseppe Cicotti is a Research Fellow at the Institute for High-Performance Computing and Networking of the National Research Council of Italy. After receiving his MSc in Computer Science at the University of Naples "Federico II", he worked 5 years in Industry as a Software Quality Engineer. In 2014 he obtained his Ph.D. in Information Engineering at the University of Naples "Parthenope" with a thesis in the field of Runtime Verification and Quantitative Analysis for Dynamic Service-Centric Systems. His current research is focused on Formal Methods and Probabilistic Risk Assessment techniques applied to Medical Software Systems to evaluate and control quality of high-regulated software.

Luigi Coppolino is currently an Assistant Professor at the University of Naples Parthenope, Italy. His research activity mainly focuses on dependability of computing systems, critical infrastructure protection, and information security. In 2005, he was in Australia, Sydney, at the University of New South Wales, for a joint research project – between Australia and Italy – on security requirements of Telemedicine systems. He has several scientific publications and was/is principal investigator of many European Commission funded research projects.

Salvatore D'Antonio is currently an Assistant Professor at the Department of Engineering of the University of Naples "Parthenope", in Italy. He is an expert in network monitoring, intrusion detection systems, and critical infrastructure protection. He was the Technical Coordinator of the FP7 EU INTERSECTION project and the Project Coordinator of the INSPIRE and the INSPIRE-INCO projects. He has several scientific publications and actively participates to IETF standardization activities.

Luigi Romano is currently a Full Professor at the University of Naples "Parthenope" in Italy. He does research in the field of system security and dependability with the Fault and Intrusion Tolerant Networked SystemS (FITNESS) group (http://www.fitnesslab.eu/). He was a Visiting Researcher at the Center for Reliable and High-Performance Computing of the University of Illinois at Urbana-Champaign. He has worked extensively as a consultant for industry leaders in the field of safety critical computer systems design and evaluation in Europe and the US. He was a member of the ENISA expert group on Priorities of Research On Current and Emerging Network Technologies (PROCENT). He is the Chair of the Cyber Security technology area of the SERIT (Security Research in Italy) platform. He is also one of the co-founders of a start-up company that is very active in R&D.