A Hybrid Framework for Web Services Reliability and Performance Assessment

Antonio Guerriero*, Raffaela Mirandola†, Roberto Pietrantuono*, Stefano Russo*
* DIETI - Università degli Studi di Napoli Federico II
Via Claudio 21, 80125, Napoli, Italy
{antonio.guerriero, roberto.pietrantuono, stefano.russo}@unina.it
† DEIB - Politecnico di Milano
Via Golgi 42, 20133 Milano, Italy
raffaela.mirandola@polimi.it

***Abstract*—We present a framework for assessing operational reliability and performance of Web Services. The framework, named WS-REPAS, is hybrid in that it combines a modelling approach, based on Discrete Time Markov Chains (DTMC), with monitoring and *in vivo* testing of the service under assessment. Through the *passive* observation of the software in operation, field data are gathered and used to continuously update values of the parameters of the service DTMC model; changes in the service provisioning, or in the way it is used, trigger an *active* strategy, which executes proper testing sessions, ultimately yielding faithful estimates of the current service reliability and/or performance. We illustrate the framework and the automated support it provides. We show how it works describing experiments with a Web service publicly available in the Github repository.**

***Index Terms*—Web services, Reliability assessment, Performance test, *In vivo* testing.**

I. Introduction

After about twenty years since their appearance, Web Services (WS) are probably still the most popular, standardized and technologically supported model for building distributed service-oriented applications. Two key Quality of Service (QoS) attributes for WS are reliability and performance: this paper addresses the problem of their assessment, namely, the engineering task of quantitatively evaluating the reliability and performance of a web service in its operating conditions.

This engineering problem is relevant in several situations: examples are the selection of a WS, the search and QoS-based ranking of functionally equivalent WS, or the composition of atomic services into a new composite WS [1]. We cope with the problem from the point of view of site operation engineers. Typically, they are required to compute QoS metrics of web services in quasi-production or in actual production environments: for instance, *site reliability engineers* at Google are "responsible to quantify confidence in the systems they maintain" [2]; similarly, quality assurance teams in *DevOps* cycles or in *Continuous Integration* practices are in charge of measuring Key Performance Indicators at *quality gates* [3].

With the goal of assessing WS reliability and performance, this paper proposes WS-REPAS, a hybrid engineering framework combining a modelling approach, based on Discrete Time Markov Chains (DTMC), with monitoring and *in vivo* testing of the service under assessment. The combination of off-line analysis techniques with run-time mechanisms for continuous verification is advocated in [4]. In WS-REPAS, field data are gathered through *passive* observations of the service in operation, and used to continuously update parameters of the service DTMC model. When changes occur in the way the service is provisioned, or in the way it is used, an *active* strategy is triggered, which executes proper testing sessions. WS-REPAS is meant for practitioners, such as site operation engineers, who can thus leverage monitoring data, usually available from WS infrastructure facilities, to feed automated analysis and testing activities, to finally produce accurate estimates of the reliability and/or performance of a Web service.

The rest of the paper is organized as follows. Section II provides some background and discusses related work. Section III gives an overview of WS-REPAS, providing background information on the modelling and testing techniques it combines. WS-REPAS is detailed in Section IV, while the automation of its key steps is discussed in Section V. Section VI presents the case study. Section VII provides concluding remarks.

II. Background and related work

There have been several initial attempts to define WS reliability. Zhang and Zhang considered it as a combination of correctness, fault tolerance, testability, interoperability, availability, and performance [5]. Zhao *et al.* viewed it, in a customer's perspective, as a function of availability and correctness [6]. Nowadays, it is usual to define WS reliability as the probability of failure-free operation under specified conditions for a specified time, similarly as for software products.[1] However, as WS offer services on demand, their reliability is often computed or estimated using a more pragmatic metric, namely as percentage of failing demands [8]. For instance, as stated in [6], "a web service that exhibits 99.7% reliability suggests that the service will fail at most three times out of one thousand attempts". We adopt this approach, expressing WS reliability in terms of the *probability*

[1] According to the IEEE Recommended Practice on Software Reliability, "*software reliability predictions are a measure of the probability that the software will perform without failure over a specific interval, under specified conditions*" [7].

of failure on demand (PFD), a discrete user-oriented metric of the above usual reliability concept. We focus specifically on *operational reliability*, i.e., the reliability of a Web service actually observed during operation.

The literature on the assessment and/or prediction of the QoS of WS is rather large. Often, the QoS metrics of individual WS are *assessed* based on historical data, so as to *predict* the QoS of a composition of WSs. A common goal of such assessment/prediction task is to support the optimal selection of services in a composition [1]. A widely explored strategy to exploit historical data is *collaborative filtering*. As the QoS of a WS depends on its implementation but also on the way it is used, for every *user-service* couple there is a specific "value" of QoS. Collaborative filters exploit such user-service QoS pairs to best combine several services in a composition. Two main collaborative filtering approaches exist:

- *Memory-based (or Neighbourhood-based)*: the QoS of missing user-service couples is predicted by exploiting users- or services-similarity. Zheng *et al.* propose a collaborative approach to predict the WS reliability for the current user, without requiring actual WS invocations, by employing the past failure data for similar users [9]. Zhang *et al.* use collaborative filtering for predicting the QoS of WS to make recommendations, based on past experiences of service users [10].
- *Model-based*: the missing QoS values are predicted considering complex relationships extracted from data, by exploiting mining and machine-learning techniques. Silic *et al.* present CLUS, a model for reliability prediction of atomic WSs, which estimates the reliability for an ongoing service invocation by aggregating data of past invocations using the *K-means* clustering algorithm [11].

In some studies, like the one by Silic *et al.*, a third class of *collaborative filtering* approaches can be defined as *hybrid*. This category includes techniques that use different combinations of collaborative filtering approaches. The work by Zhang *et al.* also presents a collaborative filtering technique in which a user-based and an item-based approach are combined together [10]. Most of these works target reliability as QoS attribute of interest; they represent the most common way to passively exploit data from monitoring to derive an estimate.

Testing is not a common practice to assess QoS attributes of WS, with the notable exception of Google's Site Reliability Engineering (see [2], Ch. 17). Indeed, testing is typically used for *fault detection* purposes at a pre-release stage or on field for enabling fault-free compositions. Few works consider testing to perform WSs composition based on the assessment of some QoS attribute [12]–[16]. Zhang proposes an approach based on dynamic testing to determine whether a WS can be integrated in a configuration while ensuring a suitable degree of reliability [12]. Tsai *et al.* propose group testing to evaluate the reliability of both atomic and composite WS; majority voting is used as an oracle for tests [13]. Di Penta *et al.* present a technique and a tool to allow users to run test suites against a service to evaluate functional and non-functional requirements [14]. Ali *et al.* propose an extensible framework to assess service composition toward a more trustworthy and reliable service ecosystem [15]. Zhu *et al.* define CARP, a black-box strategy to perform a context-aware reliability prediction of WSs [16]. CARP includes an offline step to train the context-aware reliability model from historical invocation data, followed by an online step to support on-demand reliability predictions for ongoing service invocations.

Few researchers tried to combine monitoring with testing to assess a QoS attribute of interest. Oriol *et al.* propose SALMon, a versatile service monitoring framework, that provides different strategies to get the QoS by combining passive monitoring and online testing strategies [17]. This is a general framework that do not focus on specific (monitoring-based or testing) strategy for the assessment, nor on specific QoS attributes, but enables different monitoring or testing techniques and different QoS attribute to be integrated transparently. Sammodi *et al.* introduce a framework and a prototypical implementation, based on SALMon, that exploits synergies between monitoring, online testing and quality prediction [18]. Some more details are given on test cases selection, where the strategy aims at minimizing the number of online tests performed to obtain a better coverage of service executions.

Our solution falls into this category. The monitoring-testing combination features a non-intrusive monitoring-based (black-box) estimate built upon a Bayesian model, a testing-based estimate (with high statistical confidence yet with minimum number of tests), and a strategy to continuously combine both estimates seamlessly for improved accuracy.

III. Overview

The WS-REPAS framework combines the KAMI model-based approach with an *in vivo* adaptive testing technique exploiting monitoring data for runtime reliability assessment.

KAMI is a framework (shown in Figure 1) conceived for run-time modelling of service-based systems [19]. It focuses on non-functional models which are typically dependent on (numerical) parameters that are: *i)* provided *a priori* by domain experts, or *ii)* extracted from other similar systems.

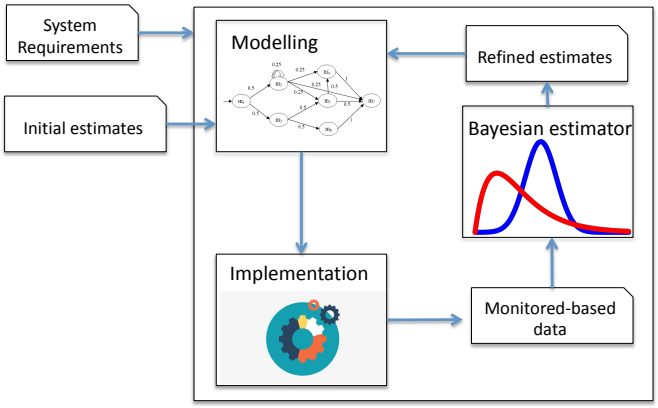

Fig. 1. The KAMI approach (adapted from [19]).

Starting from the system requirements, KAMI defines a non-functional model; the values of the model parameters are initially defined using estimates of the expected behavior of the system. This initial (imprecise or even incorrect) knowledge is called *"a priori knowledge"*. When the system is deployed, the framework collects monitoring data which correspond to model elements. For example, if the adopted model is a DTMC, the collected data are the ones representing transitions among states of the model (which are directly related to the system *usage profile*) associated with their *execution time* or *failure probability*, depending if the QoS of interest is performance or reliability. For instance, in service-based systems such events are service invocations associated with their response time and/or failure probability.

These data represent the *"a posteriori knowledge"* engineers have about the system being modeled; data feed a Bayesian estimator as defined in [19], in charge of producing new estimates of the model parameters. These updated models provide a more accurate representation of the current behavior of the system, and allow engineers to automatically check conformance to requirements while the system is running.

In this perspective, the accuracy of the initial values adopted as *a priori* knowledge does not affect the effectiveness of estimates of model parameters. Indeed, KAMI can also be useful in case of greenfield projects, where no *a priori* knowledge exists. In these case, random values might be adopted as *a priori* knowledge and the Bayesian estimation produces the correct estimates even if requiring more run-time data.

Regardless of the specific modelling formalism used, this approach is a *passive* strategy, since the assessment is based just on the observation of the system in operation. Passive strategies have the advantage of coming at low cost (monitoring is the main cost it incurs, provided that the model is simple enough to be solved/updated); however, they are limited in terms of accuracy especially in an evolving context for (at least) three reasons: *i)* due to the passive nature of monitoring, the demand space might be not explored adequately (e.g., never exercising failing demands, or never using a subdomain), yielding large-variance and inaccurate estimates; *ii)* in a highly-dynamic environments (e.g., service-based applications), both the usage and the failure probability are expected to change quickly; *iii)* even under a stable usage profile and failing behaviour, the estimate of a pure passive approach needs time to converge to the true one, as, regardless of the specific model, a certain number of observations are needed for confidently state a QoS value. What can happen is that the technique could never converge because of sudden changes of the profile, hence yielding a constantly old and inaccurate estimate of the QoS of interest.

WS-REPAS complements *passive* information with an *active* strategy, combining data coming from monitoring and data generated by properly designed online testing sessions. This allows to actively "spot" those demands more informative about the current reliability, yielding small-variance (i.e., high-confidence) estimates, at the cost of running a testing session when needed. An efficient testing algorithm is exploited to have confident estimates with few tests. Figure 2 outlines the proposed framework, which is detailed in the next sections.

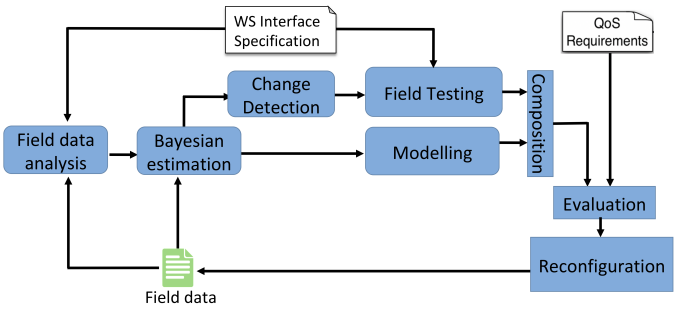

Fig. 2. The WS-REPAS hybrid modelling/testing framework

IV. WS-REPAS

A. Modelling

Modelling aims at providing a suitable description of the system under assessment. Several models can be leveraged from the literature of performance, reliability and availability assessment [19], the most common ones being Markovian models, Queueing Networks, (Stochastic) Petri Nets or combinatorial models. We rely on Discrete Time Markov Chains.

A DTMC is characterized by a set of states and transition probabilities between them. The chain evolves in discrete steps. The one-step *transition probability matrix* $M = [m_{i,j}]$ is a stochastic matrix wherein each $m_{i,j}$ is the probability of going to state j at step n from state i at step $(n-1)$. We use *irreducible* DTMCs (i.e., with no absorbing state), as they are well suited for continuously running applications like Web services. States model service invocations, and transitions represent sequences of invocations.[2] Specifically, depending on the preferred granularity and on the WS application size, a state can represent *i)* the invocation of a service, namely, of any of its method (*service-level*), or *ii)* the invocation a specific method of a service (*method-level*), or *iii)* the invocation of a specific method of a service with inputs belonging to a specific partition of the method's input space (*partition-level*). In the following, we assume the latter one, namely the finest granularity. The information, for all the cases, is retrieved from the WS specification, as detailed later in the paper.

The state-dependent **probability vector** $\mathbf{p}(n)$ at step n of a DTMC, representing the probabilities of being in each state at time n, is fully determined by the transition matrix M, since: $\mathbf{m}(n) = \mathbf{m}(0)\ M^n$, with $\mathbf{m}(0)$ being the initial probability vector. As $n \to \infty$, $m_{i,j}$ values become independent of both n and i: all the rows of the matrix M converge toward a common limit v_j. The set of v_j values is the *steady-state probability vector* $\mathbf{v}$, namely the probability of being in state j as $n \to \infty$: $v_j = \sum_i v_i m_{i,j}$. Its value can be obtained starting from $\mathbf{v}^{(0)} = \mathbf{m}(0)$ and then iteratively applying $\mathbf{v}^{(n+1)}$

[2]The representation is suited for capturing state dependencies due to shared data (which are present both in composite WSs, SOAP WSs and in many REST services, supposed to be, in principle, stateless) or state dependencies caused by the environment, such as hardware, external systems, etc.

= $\mathbf{v}^{(n)}M$ until convergence is reached [20]. For computing the expected reliability or performance of a service, the j-th state is associated with a reward r_j (in our case, the *probability of failure on demand* – PFD - for (un)reliability, and the *throughput* for performance). From the steady-state probability vector, the steady-state reward Z is given by: $E[Z] = \sum_j v_j r_j$. Transient rewards may be of interest too.

B. Bayesian estimation

Data gathered from monitoring are exploited to update the DTMC's parameters by a Bayesian approach. Specifically, we update the transition probability matrix M (i.e., the usage probability) and the rewards (PFD and throughput).

For the update of the matrix M, we assume that information about the occurrence of every transition from state *i* to state *j* in the DTMC can be collected through run time monitoring.

In a Bayesian perspective the transition matrix M is a random matrix and the statistical problem of updating each $m_{i,j}$, using run-time data, corresponds to updating the *prior distribution* of M (depending on $\mathbf{m}(\mathbf{0})$) computing a *posterior* conditional probability given the run-time data.

Then the *posterior* distribution leads to a new estimate of M. Hence, the Bayesian solution of updating M requires a statistical model and a prior distribution of M. Regarding the prior distribution of M, we assume statistical independence among its rows and model each row $(m_{i,1}, \ldots, m_{i,k})$ with a *Dirichlet Distribution*.[3] It yields a posterior distribution, which is still a Dirichlet, from which we derive an updated transition matrix through an updating rule that can be expressed as the weighted sum of our initial estimate and of the knowledge we extract from run-time data. The weights can be defined in several ways and they quantify the confidence we have on the *a priori knowledge* with respect to run time data.

In WS-REPAS, the weight assigned to the prior distribution is set to zero each time a testing session is triggered, namely each time a significant change in the reliability or performance estimate has been detected (e.g., because of the changed profile or services' PFD or throughput), as detailed in the next Section. In fact, in such a case, the far history is uninformative and can slow down the convergence.

A complete description of the mathematical steps involved in this process is beyond the scope of this paper, interested reader can find detailed information in [19].

As for rewards, the PFD values are updated by a Beta distribution. Let us denote as f_i the PFD associated with the i-th partition of a service method. Requests are characterized by a binary outcome: they can be successful or not. We are interested in the number of failures over N_i demands, which is well captured by a binomial distribution with parameters N_i and f_i. The failure probability f_i in a service-based system context is likely to be not a fixed value, but it can change over time. This is suitably represented as a random variable with Beta distribution $Beta(a; b)$, which well represents proportions in binomial processes, coming to the well-known Beta-Binomial distribution. Then, the estimate of f_i is given by: $\hat{f}_i = E[f_i] = \frac{a_i}{a_i+b_i}$.

With a similar reasoning, we update the throughput using a *Poisson-gamma* distribution, which uses the Gamma as conjugate distribution for the rate parameter of the Poisson process. In particular: let $t_i > 0$ be the throughput of method i that we want to update. The $Poisson(t_i)$ distribution models the number of responses that have been received in a given time interval; as before, the rate t_i changes over time and can be modelled as a $Gamma(u, v)$ distribution, with expected value equal to: $\hat{t_i} = E[t_i] = \frac{u_i}{v_i}$.

Thus, the overall model consists of an m-variate Dirichlet distribution to capture the uncertainty about the operational profile, of n univariate Beta and Gamma distributions to capture the uncertainty about the PFD and about the throughout, respectively. These have all the nice property that the posterior preserves the form of the prior distribution, and just the parameters need to be updated as new observations are gathered. Suppose that $N_1, N_2, \ldots N_m$ new demands are done to the WS methods' partitions $1, 2, \ldots, m$, out of which $x_1, x_2, \ldots x_m$ fail and with response times $y_1, y_2, \ldots y_m$: the new distributions will be: $D(\alpha_1+N_1; \alpha_2+N_2; \ldots; \alpha_m+N_m)$ for the operational profile; $Beta_i(a_i + x_i; b_i + N_i - x_i)$; $Gamma_i(u_i + N_i; v_i + y_i)$. Thus, both $\hat{p}_i$, $\hat{f}_i$ and $\hat{t}_i$ are updated by this simple mechanism, which requires a negligible computational cost.

C. Change detection

In this step, we aim at revealing relevant changes in the usage and/or of QoS attribute of interest (e.g., PFD and/or throughput), so as to trigger testing only when a severe change happens, in which case the pure model-based estimate is no longer reliable for what said previously.

Let us first define an "observation" s_t at (discrete) time t as a request-response pair. Each time such an observation is collected, the $\hat{p}_i$, $\hat{f}_i$ and $\hat{t}_i$ values of each partition are updated as described previously – hence yielding the updated reliability and performance estimate by the DTMC solution. These are the estimates computed via just monitoring.

Let us now consider a sliding window W_t consisting of the last k observations before t ($W = \{w_{t-k}, w_{t-k+1} \ldots, w_t\}$), where: k is the window size and is set to k_0. We perform a statistical hypothesis test on the sample W_t in order to check if the reliability or performance estimate is significantly increasing (or decreasing) or not. We opt for the *Mann-Kendall* (MK) test, which is a widely-adopted non-parametric trend detection test making no assumption on the data distribution. It tests the null hypothesis that *there is no trend in data*, with a given significance level α, against the alternate hypothesis that a trend exists.[4] If, according to the MK test, there is no trend, it means that the *monitoring estimate* is "stable" and has converged to true value (with a confidence of 95%). If a

[3]The choice of the Dirichlet distribution is justified in [21], which proves that, in a multinomial model, the prediction of a future event is linear in the number of past occurrences if and only if the prior distribution is Dirichlet.

[4]We set: $\alpha = 0.05$, namely a confidence of 95%; $k_0 = 50$.

trend exists, it means that the estimate is not stable and is still converging toward the true value.

Testing is run at the current time t only when a trend is detected at time t on the window W_t – hence the current monitoring estimate is not stable – *and* no trend was detected at time t-1 on W_{t-1}. In other words, the monitoring estimate was judged as "stable" at step t-1, and now a new trend has been detected (e.g., because the true reliability/performance changed and the estimate started converging toward the new true value). This requires a new testing session to get a *testing estimate* soon, rather than waiting for the monitoring estimate to converge.

D. Field testing

Testing will complement the monitoring estimate when required. The goal is to have high-confidence estimates with as few tests as possible, by running *in vivo* tests. The proposed approach exploits the *survey sampling* theory, which enables elaborate schemes leveraging auxiliary information to drive the test case generation process [22].

Specifically, we implement a *weighted* operational testing, in which the estimate of the usage profile ($\hat{p}_i$) is "weighted" by the estimate of the reward ($\hat{f}_i$ or $\hat{t}_i$, depending on the quality attribute to estimate), in order to select test cases. This forms the testing profile, according to which tests are generated. The underlying idea is to accelerate the occurrence of failing or low-performance requests, rather than waiting for them to happen in operation (like when we use only $\hat{p}_i$), thus having the double positive effect of *i)* anticipating failures and *ii)* improving the estimate's accuracy, as demonstrated in our previous work [23], [24]. We adapt the algorithm used in our previous study, named *Microservice Adaptive Reliability Testing* ($MART$) for Microservice architecture (MSA) applications [25], customized to work for both reliability and performance testing of Web Services (namely, with both the $\hat{f}$ and $\hat{t}$ estimates). This is a without-replacement strategy, which is known to work better at the expense of trickier mathematical treatment on the estimator. To get an unbiased estimate of the expected PFD and of the expected throughout, the Thompson's estimator is exploited, which is suitable for the implemented sampling strategy [26]. Details of the algorithm are in our previous work [25].

E. Composition

This step is in charge of combining the model-based (M_e) and testing-based (T_e) estimates. To this end it implements a set of composition operators, $\otimes$ such that, the results of

$$Comp_e = M_e \otimes T_e$$

can feed the *Evaluation* module.

According to the case under exam, $\otimes$ can be instantiated in several ways. For example, $\otimes$ can be defined as the *worst case* estimate, whenever a conservative approach is the most appropriate one; or $\otimes$ could be a weighted average $w_e M_e + w_t T_e$, where the values of w_e and w_t depend on the trust associated to the estimates. For example, it could be $w_t = 1$ (and consequently $w_e = 0$) when a change is detected and until a stable phase is entered again. According the the need, $Comp_e$ could be a single value representing either performance or reliability estimates, or it could be an aggregated value combining both estimates, or it could be decomposed in a couple of values $< Comp_e(R), Comp_e(P) >$ whenever the need of dealing in a separate way with reliability and performance arises.

F. Evaluation

This module takes in input the results of the composition and the QoS requirements and implements a comparison function Φ that can trigger reconfiguration actions.

This function can be implemented in several ways according to the system of interest. For example, Φ could be a simple operation, like a comparison with a given threshold of performance and/or reliability, or it could be a check that $Comp_e$ belongs to a given range of values. Φ could also implement more complex functions like the computation of Pareto front.

G. Reconfiguration

This is in charge of implementing reconfiguration actions, triggered by the previous evaluation. For example, if the reliability estimate highlights that a specific service or method (represented by the expected reward in the state representing that service/method/partition) has a high PFD, then the action can be an update of the service to improve the PFD or a fault tolerance action. Similarly, if a service/method exhibits a high expected throughput because of a high load (represented by high transition probability values toward the service/method/partition), then a load-balancing action, such as redirecting requests, can be implemented, which result in changing the transition probabilities, solving the bottleneck. The implementation of specific reconfiguration actions is left to future work.

V. Tool support

This Section discusses the automated support to the main steps of the proposed WS-REPAS framework.

A. WS input partitioning and test cases generation

The partitioning of the input domain is performed starting from a specification of the interfaces in the *swagger* format (the *WS Interface Specification* in Figure 2). It consists of a JSON documentation of the application, with a huge number

TABLE I
List of JSON attributes

host	hostname and port number
paths	URL in the host that identify the resource
method	http methods allowed for the URL
parameters	list of parameters specified in the URL or in the payload
name	parameter name
in	location of the parameter (path or body)
type	parameter type (Integer, String, ...)
maximum	maximum value of the parameter, or specific symbol/character
minimum	minimum value of the parameter

of keys to describe the host, the methods (URI templates), the parameters, the expected responses, and so on. Starting from this one we derive a simplified JSON description of the interfaces, where we consider the attributes described in Table I. The attributes in bold are mandatory; those in Italic are optional. A method can have no *parameters*; if a parameter is specified, *name*, *in*, and *type* must be defined (*maximum* and *minimum* values are optional). An example of the extracted JSON specification of the methods follows.

```
{
  "host": "example.com:123",
    "paths":{
      "/foo":{
        "get":{
          "parameters":[]
        }
      },
      "/foo/{op1}":{
        "get":{
          "parameters":[{
            "name":"op1",
            "in":"path",
            "type":"Integer",
            "maximum":10,
            "minimum":0
          }]
      }
    }
  ...
}
```

Starting from this specification, *partitioning* is performed defining equivalence classes of the input values. If maximum and/or minimum values are not specified, a default partitioning is considered based on the type of specified parameters. Each partition is encoded in a data structure called *test frame*, with associated the estimates of the operational profile, PFD and throughput (namely, $\hat{p}_i$, $\hat{f}_i$ and $\hat{t}_i$, respecitvely) used by the test case generation algorithm previously described.

B. Field data analysis

Execution data about the WS under test are gathered by a monitoring infrastructure and processed by a *parser* (see Figure 2) to extract the information necessary for change detection and for the testing strategy. This takes in input a couple request-response, which is obtained by the monitoring infrastructure. The parser derives the partition (i.e., the test frame) the input belongs to, hence feeding the Bayesian estimation module, which updates the information as specified in Subsection IV-B. This allows obtaining a fresh estimate by the defined models and to improve the testing algorithm performance.

C. Change detection

The change detection is performed by the *Change Detector* component in Figure 2. This collects the latest estimates of reliability, and performs the *Mann-Kendall* test using the *Kendall library* of *R* statistical computing language.[5]

[5]https://CRAN.R-project.org/package=Kendall.

D. Tests execution

After every request, we have an estimate of reliability. This estimate is chosen between the one deriving from the model (*Modelling* in Figure 2) and the one from testing (*Testing-based estimator* in Figure 2). Different ways can be conceived to combine or chose between the estimates (*composition* in Figure 2): in our current implementation, the policy is as follows: testing is run only when the monitoring estimate has a trend (like explained in Section IV-C); in such a case, the testing estimate is expected to be more accurate, as monitoring is not stable yet: hence we consider the last testing estimate for all the subsequent time steps until the monitoring estimates converges or the difference between the two becomes negligible (i.e., $< \epsilon = 0.001$).

VI. Case study

For illustrative purpose, we evaluated WS-REPAS on a RESTful service for managing products feature models (namely, compact representations of the features of products in a Software Product Line), called *Feature Service* (FS)[6], having 7 methods, 1,712 lines of code mostly in Java and SQL. The purpose of the evaluation is to show the feasibility of our approach in both a stable and a variable profile scenario.

A. Test infrastructure

The test infrastructure for the experiments consists of the following components:

- *Workload generator*: it emulates clients of the application, executing demands according to the true profile; after a certain number of observations, the profile and the failure probabilities may be changed so as to emulate a variable profile, e.g. due to an upgrade of a service;
- *Monitor*: it performs the monitoring of the application usage, feeding the WS-REPAS Engine. To this aim, we use the *MetroFunnel* monitoring tool tailored for our purpose, developed in our research group;[7]
- *Bayesian estimator*: that exploit the information obtained by monitoring to update the estimated profile, the failure probability and the estimated throughput;
- *Test generator*: it implements the testing algorithm to select the test frames according to the operational profiles weighted by the failure probability or throughput estimates to get reliability and performance estimates based on test results;
- *Estimator*: it assesses reliability and performance according to the sampling and modelling strategies adopted.

B. Setup and execution

Two experimental scenarios are run to show the feasibility of our approach. The first one is performed considering 900 observations under a stable operational profile, so as to check for the convergence of the estimates to the true values. For the second one, we consider 900 observations, split in three sets.

[6]*https://github.com/JavierMF/features-service*

[7]MetroFunnel is available at: https://github.com/dessertlab/MetroFunnel.

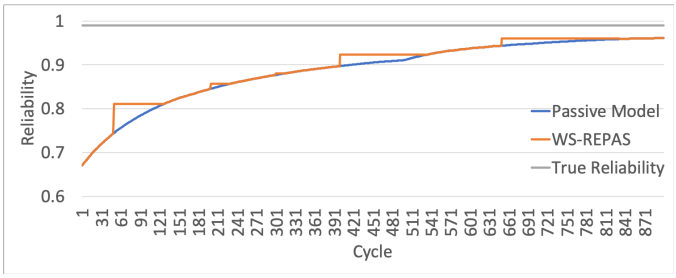

Fig. 3. Example of WS-REPAS execution to estimate the reliability in a static operational profile condition, compared to the modelling approach without testing

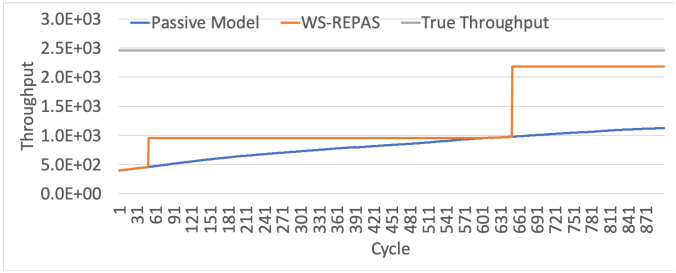

Fig. 4. Example of WS-REPAS execution to estimate the throughput in a static operational profile condition, compared to the modelling approach without testing

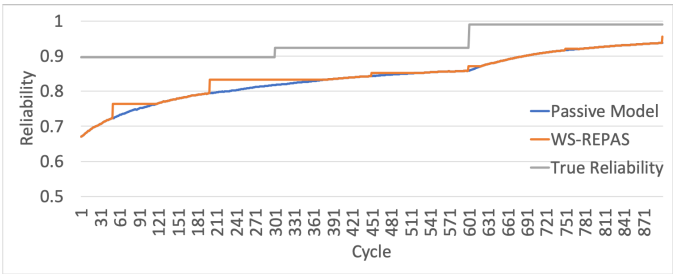

Fig. 5. Example of WS-REPAS execution to estimate the reliability in a variable operational profile condition, compared to the modelling approach without testing

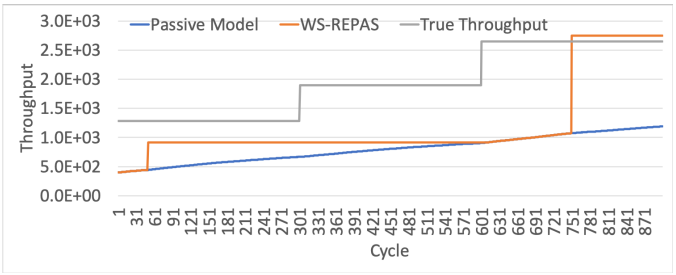

Fig. 6. Example of WS-REPAS execution to estimate the throughput in a variable operational profile condition, compared to the modelling approach without testing

Each set is related to a different true operational profile, hence emulating a scenario in which the usage of the WS changes significantly during operation.

The test is configured considering the 80% of the total number of test frames as budget. This value corresponds to 130 tests per execution.

C. Results

Figures 3 and 4 show that WS-REPAS converges to the true reliability and performance. where the performance of WS-REPAS are compared to the pure modelling approach, namely, without the support of testing. In both cases, the support of testing improves the estimate of the modelling approach, producing values that are more representative of the true information both in terms of reliability and of throughput.

Figures 5 and 6 report the result under the variable profile scenario. At 300 and 600 requests, the true profile changes: both Figures show the ability of WS-REPAS to adapt its estimates to the variability of the true profile, following the changes of the operational profile. In the case of throughput, this is even more evident, since the testing estimate is able to get much closer to the true value than the modelling approach; the latter has a very slow convergence, and would clearly require much more time to converge.

Two important remarks are: *i)* the testing-supported estimate is able to "anticipate" the true value, sometimes by a large amount of time steps, which can be useful to take preventive actions before failure or performance loss is experienced by end users; *ii)* this, of course, has a cost: in the shown example case study, it amounts to 5 testing sessions for the reliability assessment (Figures 3 and 5) and 2 testing sessions for the performance assessment (Figures 4 and 6).

In the future, our aim is to precisely assess the trade-off between the gain obtained (in terms of accuracy and of "anticipation") and the cost incurred (e.g., trying the approach with different testing budget and/or different testing algorithms; trying different policies to minimize the number of times that testing is triggered). Moreover, the problem of test isolation is outside the scope of this work, but our aim is to address the challenges and side-effects brought by running *in vivo* tests in operational instances.

VII. Conclusion

The quantitative evaluation of the reliability and performance of Web services is an important engineering activity, especially in modern DevOps and continuous integration practices. In such contexts, we believe this activity can be carried out accurately and effectively through the seamless combination of modelling and testing techniques, leveraging field data which are typically made available to operation engineers by monitoring infrastructures.

In this papers, we have presented WS-REPAS, a framework which combines modelling by means of Discrete Time Markov Chains and an active in vivo black-box testing strategy, providing confident estimates with relatively few runtime tests. The framework is automated to a large extent; in particular, test cases for the Web service under test are automatically generated from a specification of its interfaces, based on a partitioning of its input arguments.

ACKNOWLEDGMENT

This work has been supported by the PRIN 2015 project "GAUSS" funded by MIUR (Grant n. 2015KWREMX_002).

REFERENCES

[1] Y. Zhang and M. Lyu, *QoS Prediction in Cloud and Service Computing: Approaches and Applications*, ser. SpringerBriefs in Computer Science. Springer, 2017.

[2] B. Beyer, C. Jones, J. Petoff, and N. R. Murphy, Eds., *Site Reliability Engineering - How Google Runs Production Systems*. O'Reilly Media, Inc., 2016.

[3] R. Pietrantuono, A. Bertolino, G. De Angelis, B. Miranda, and S. Russo, "Towards Continuous Software Reliability Testing in DevOps," in *IEEE/ACM 14th Int. Workshop on Automation of Software Test (AST)*. IEEE, 2019, pp. 21–27.

[4] A. Filieri, C. Ghezzi, R. Mirandola, and G. Tamburrelli, "Conquering Complexity via Seamless Integration of Design-Time and Run-Time Verification," in *Conquering Complexity*, M. Hinchey and L. Coyle, Eds. Springer, 2012, pp. 253–275.

[5] J. Zhang and L. Zhang, "Criteria Analysis and Validation of the Reliability of Web Services-oriented Systems," in *IEEE Int. Conference on Web Services (ICWS'05)*. IEEE, 2005.

[6] S. Zhao, X. Lu, X. Zhou, T. Zhang, and J. Xue, "A Reliability Model For Web Services - From The Consumers' Perspective," in *Int. Conference on Computer Science and Service System (CSSS)*. IEEE, 2011, pp. 91–94.

[7] IEEE, "IEEE Recommended Practice on Software Reliability," *IEEE Std 1633-2016*, 2017.

[8] H. Zo, D. Nazareth, and H. Jain, "Measuring Reliability of Applications Composed of Web Services," in *40th Annual Hawaii Int. Conference on System Sciences (HICSS)*, 2007, p. 278c.

[9] Z. Zheng and M. R. Lyu, "Collaborative reliability prediction of service-oriented systems," in *2010 ACM/IEEE 32nd Int. Conference on Software Engineering*, vol. 1, 2010, pp. 35–44.

[10] Z. Zheng, H. Ma, M. R. Lyu, and I. King, "QoS-Aware Web Service Recommendation by Collaborative Filtering," *IEEE Transactions on Services Computing*, vol. 4, no. 2, pp. 140–152, April 2011.

[11] M. Silic, G. Delac, and S. Srbljic, "Prediction of Atomic Web Services Reliability for QoS-Aware Recommendation," *IEEE Transactions on Services Computing*, vol. 8, no. 3, pp. 425–438, 2015.

[12] J. Zhang, "An approach to facilitate reliability testing of web services components," in *15th Int. Symposium on Software Reliability Engineering*. IEEE, 2004, pp. 210–218.

[13] W.-T. Tsai, D. Zhang, Y. Chen, H. Huang, R. Paul, and N. Liao, "A software reliability model for web services," in *Proc. of the Eigtht IASTED International Conference on Software Engineering and Applications*, M. H. Hamza, Ed., 2004, pp. 144–149.

[14] M. Di Penta, M. Bruno, G. Esposito, V. Mazza, and G. Canfora, "Web Services Regression Testing," in *Test and Analysis of Web Services*, L. Baresi and E. Nitto, Eds. Springer, 2007, ch. 8, pp. 205–234.

[15] M. Ali, F. De Angelis, D. Fanì, A. Bertolino, G. De Angelis, and A. Polini, "An extensible framework for online testing of choreographed services," *Computer*, vol. 47, no. 2, pp. 23–29, 2014.

[16] J. Zhu, P. He, Q. Xie, Z. Zheng, and M. R. Lyu, "CARP: Context-Aware Reliability Prediction of Black-Box Web Services," in *2017 IEEE Int. Conference on Web Services (ICWS)*. IEEE, 2017, pp. 17–24.

[17] M. Oriol, X. Franch, and J. Marco, "Monitoring the Service-based System Lifecycle with SALMon," *Expert Systems with Applications*, vol. 42, no. 19, pp. 6507–6521, 2015.

[18] O. Sammodi, A. Metzger, X. Franch, M. Oriol, J. Marco, and K. Pohl, "Usage-based online testing for proactive adaptation of service-based applications," in *2011 IEEE 35th Annual Computer Software and Applications Conference*. IEEE, 2011, pp. 582–587.

[19] I. Epifani, C. Ghezzi, R. Mirandola, and G. Tamburrelli, "Model Evolution by Run-Time Parameter Adaptation," in *31st Int. Conference on Software Engineering (ICSE)*. IEEE, 2009, pp. 111–121.

[20] G. Bolch, S. Greiner, H. de Meer, and K. Trivedi, *Queueing networks and Markov chains: modeling and performance evaluation with computer science applications*. New York, NY, USA: Wiley, 1998.

[21] P. Diaconis and D. Ylvisaker, "Conjugate priors for exponential families," *Ann. Statist*, vol. 7, no. 2, pp. 269–281, 1979.

[22] R. Pietrantuono and S. Russo, "On Adaptive Sampling-Based Testing for Software Reliability Assessment," in *27th Int. Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2016, pp. 1–11.

[23] R. Pietrantuono and S. Russo, "Probabilistic sampling-based testing for accelerated reliability assessment," in *IEEE Int. Conference on Software Quality, Reliability and Security (QRS)*. IEEE, 2018, pp. 35–46.

[24] D. Cotroneo, R. Pietrantuono, and S. Russo, "RELAI testing: A technique to assess and improve software reliability," *IEEE Transactions on Software Engineering*, vol. 42, no. 5, pp. 452–475, 2016.

[25] R. Pietrantuono, S. Russo, and A. Guerriero, "Run-time Reliability Estimation of Microservice Architectures," in *29th Int. Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2018, pp. 25–35.

[26] D. G. Horvitz and D. J. Thompson, "A generalization of sampling without replacement from a finite universe," *Journal of the American Statistical Association*, vol. 47, no. 260, pp. pp. 663–685, 1952.