Runtime Prediction of Software Service Availability

Davide Lorenzoli, George Spanoudakis
Department of Computing, City University London, London, UK

Abstract This paper presents a prediction model for software service availability measured by the mean-time-to-repair (MTTR) and mean-time-to-failure (MTTF) of a service. The prediction model is based on the experimental identification of probability distribution functions for variables that affect MTTR/MTTF and has been implemented using a framework that we have developed to support monitoring and prediction of quality-of-service properties, called EVEREST+. An initial experimental evaluation of the model is presented in the paper.

Keywords: Run-time QoS Prediction, Software Services

1 Introduction

The monitoring of quality-of-service (QoS) properties of software services at runtime is supported by several approaches (see [2] for a survey). Most of these approaches, however, can only detect violations of QoS properties after they have occurred without being able to predict them. This is a significant limitation as the capability to predict violations of QoS properties at runtime is important for the dynamic and proactive adaptation of both the services and the systems that use them. The prediction of QoS properties violations is a challenging problem. This is because there is often a lack of models (e.g., service behavioural models, deployment infrastructure models) that would enable a white-box approach to prediction based on analytic techniques (e.g. [21]). Hence, QoS property prediction has often to rely on black-box techniques analysing QoS service data collected at runtime.

In this paper, we present a runtime prediction model for a key QoS property of software services, namely software service availability. The model is based on the prediction of two measures of service availability: the mean-time-to-failure (MTTF) and mean-time-to-repair (MTTR) of a software service, defined as the average up and down time in the operational life of a service, respectively [19][20]. Our MTTR/MTTF prediction model is based on the experimental identification of probability distribution functions for the variables that affect these two measures.

The MTTR/MTTF model has been implemented using an integrated framework that we have developed to support monitoring and prediction of QoS properties of software services expressed in service level agreements (SLAs), called EVEREST+. EVEREST+ enables the monitoring of QoS and behavioural service properties expressed in Event Calculus, and the development and automatic maintenance of runtime models for estimating the probability of potential violation/satisfaction of constraints over such properties (i.e., prediction models).

The development of prediction models in EVEREST+ is based on prediction specifications determining: (a) the key variables underpinning the QoS properties that need to be predicted, (b) the patterns of events that need to be monitored in order to collect runtime data about these variables, and (c) the computations that need to be performed to generate predictions for the properties. Based on these specifications, EVEREST+ can infer automatically the probability distribution functions of the variables that underpin the QoS properties of interest at runtime using monitoring data, and uses these functions to calculate the probability of future violations of the QoS properties.

The rest of this paper is structured as follows. In Sect. 2, we describe the prediction model for software service MTTR and MTTF. In Sect. 3, we describe how the MTTR/MTTF models are realised in EVEREST+. In Sect. 4, we present the results of an initial experimental evaluation of the prediction model. Finally, in Sect. 5 we discuss related work and in Sect. 6, we present concluding remarks and directions for future work.

2 Prediction Model

2.1 Overview

A QoS property prediction in our approach is the computation of the probability that the QoS property will violate a constraint at some future time point \(t_0\), given a request for it received at a time point \(t_r\). Figure 1 illustrates this general formulation of the prediction problem. In particular, \(t_r\) in the figure is the time point at which a prediction for a property QoS is requested; \(t_0\) is the time point in the future that the prediction is required for; \(p\) is the prediction window (i.e., \(p = t_0 - t_r\)); \(N\) is the number of QoS values observed between \(t_r\) and \(t_0\); \(Y\) is the number of future QoS property values that are expected between \(t_r\) and \(t_0\); \(QoS_r\) is the value of the observed QoS property at the time point \(t_r\); and \(QoS_0\) is the value of the predicted QoS property at the time point \(t_0\).

![Figure 1. Prediction framework common definitions](image)

Given this generic formulation, the computation of the probability of violating a QoS property constraint is based on estimating the probability of occurrence of different values of variables that affect the QoS property in the prediction window, and can make it violate the constraint. The
probabilities of the values of the underpinning variables are determined by finding the probability distribution functions (PDFs) that have the best fit to sets of historic values of these variables.

2.2 Prediction Model for Software Services MTTR

The overall approach outlined above has been used to develop the prediction model for the mean-time-to-repair (MTTR) and mean-time-to-failure (MTTF) of a service. More specifically, the MTTR of a software service is defined as the average time from a failure of a service to respond to an operation call until it restarts responding to operation calls normally. MTTR needs to be bounded to ensure the timely reactivation of a service after periods of unavailability. In an SLA, this would be typically specified as a boundary constraint of the form: MTTR ≤ K where K is a constant time measure.

The estimation of the probability of violating the boundary constraint MTTR ≤ K at a future time point t c is based on identifying the probability distribution functions of two variables: (1) the MTTR of the service of concern, and (2) the time between non-served calls of service operations (i.e., service failures) that occur in a period during which a service has been available (referred to as time-to-failure or “TTF” henceforth). MTTR and TTF values correspond to the periods in the operational life of a service shown in Figure 2. More specifically, MTTR is computed as the average of TTR values, i.e., the time difference between the timestamp of the first served call of a service following a period of unavailability and the timestamp of the initial non served call (NS Call) of the service that initiated this period. TTF is the difference between the timestamps of two NS calls of the service that initiate two distinct and successive periods of unavailability.

![Figure 2. TTR and TTF values](image)

Assuming that N is the number of MTTR values recorded until the time t c at which the prediction is requested, t c is the future time point for which the prediction is requested, and y is the yet unknown – number of TTF values that will be recorded during the prediction horizon p (or, equivalently, the number of cases where the service fails again following a period over which it has been available), to violate the MTTR constraint at t c the following condition must be false:

\[ \text{MTTR}_c = (N \times \text{MTTR}_c + y \times \text{MTTR}_y) / (N + y) \leq K \]  \hfill (1)

According to (1), the MTTR value at t c (i.e., MTTR_c) is computed by replacing each of the N TTR-values that have been recorded until t c by the average value MTTR_c that has been recorded until t c, and each of the y TTR-values from t c to t c by their average value MTTR_y since:

\[
\sum_{j=1}^{N} \text{TTR}_j = N \times \text{MTTR}_c \\
\sum_{j=N+1}^{N+y} \text{TTR}_j = y \times \text{MTTR}_y
\]

From formula (1), however, it can be deduced that for the MTTR constraint to be violated it must be that:

\[ \text{MTTR}_y \geq (K \times (N + y) - N \times \text{MTTR}_c) / y \]  \hfill (2)

Given (2), however, there are two factors to take into account to predict MTTR:

(a) The probability to observe y reactivations of the service in the time period \( p \) from \( t_c \) to \( t_o \), denoted as \( Pr(y) \) in the following, and

(b) The probability of having \( MTTR_y \) > \( MTTR_{crit} \) (i.e., \( Pr(MTTR_y > MTTR_{crit}) \)) where \( MTTR_{crit} = (K \times (N + y) - N \times \text{MTTR}_c) / y \).

Based on the probabilities (a) and (b), the probability to violate the constraint \( MTTR \leq K \) at the end of the time period \( p \) can be estimated approximately by the following formula:

\[
Pr \left( \bigwedge_{y \geq 1} E_y \right) = \begin{cases} 
1 - \sum_{y=1}^{N} Pr(y) \times Pr(MTTR_y \leq MTTR_{crit}), & MTTR_y > K \\
\sum_{y=1}^{N} Pr(y) \times Pr(MTTR_y > MTTR_{crit}), & MTTR_y \leq K
\end{cases}
\]  \hfill (3)

Formula (3) distinguishes two cases: (a) the case where the last recorded MTTR value at the time when the prediction is requested violates the constraint (i.e., the case where \( MTTR_y > K \)), and (b) the case where the last recorded MTTR value at the time when the prediction is requested does not violate the constraint (i.e., the case when \( MTTR_y \leq K \)). In the former case, the probability of violation is computed as the probability of not seeing a value of MTTR in period \( p \) (i.e., \( MTTR_y \)) that is sufficiently small to restore the current violation. In the second case, the probability of the violation is computed as the probability of seeing a large enough \( MTTR_y \) value (i.e., a value greater than \( MTTR_{crit} \)) that would violate the constraint. However, since the value of \( y \) is not known at \( t_c \) when the prediction is requested, formula (3) considers values of \( y \) up to an upper limit M. The value of M is determined by a condition over its probability. More specifically, M is set to the largest value that makes \( Pr(y=M) \) arbitrarily small (i.e., \( Pr(y=M) < e \) for an arbitrary small \( e \)).

To compute the probabilities \( Pr(y) \), \( Pr(MTTR_{crit} \leq MTTR_y) \), and \( Pr(MTTR_y > MTTR_y) \) we need to know the cumulative distribution functions (CDF) for the variables \( y \) and \( MTTR \), referred to as \( \text{CDF}_y \) and \( \text{CDF}_{MTTR} \) respectively. Given \( \text{CDF}_{MTTR} \), \( Pr(MTTR_{crit} > MTTR_y) \) and \( Pr(MTTR_{crit} \leq MTTR_y) \) can be computed by formulas (4) and (5) below:

\[
Pr(MTTR_{crit} > MTTR_y) = 1 - \text{CDF}_{MTTR}(MTTR_{crit}) \\
Pr(MTTR_{crit} \leq MTTR_y) = \text{CDF}_{MTTR}(MTTR_{crit})
\]

To compute \( Pr(y) \) we do not use \( \text{CDF}_y \), but the probability distribution function of TTF variable (i.e., the difference between the timestamps of two NS calls of the service that initiate two distinct and successive periods of unavailability,

\[ Pr(y) = \int_{MTTR_y}^{MTTR_{crit}} \text{PDF}(y) \, dy \]

1 The value of e is set by the requester of the prediction.
as defined earlier). This is because over a time period of $p$ time units, $TTF$ is related to $y$ as shown by the formula below:

$$TTF = p/y$$  \hfill (6)

As (6) indicates, $TTF$ is an invertible monotonic function $g$ of $y$ (i.e., $TTF = g(y) = p/y$ where $p$ is constant). Thus, the cumulative probability distribution of $y$ $CDF_y = gCDF_{TTF}$ can be computed by the formula:

$$CDF_y(y) = 1 - CDF_{TTF}(g^{-1}(y)) = 1 - CDF_{TTF}(p/y)$$  \hfill (7)

where $g^{-1}(y) = TTF = p/y$.

Furthermore, to compute $Pr(y)$, instead of considering a single value $v$ of $y$, we consider the range $\hat{y}(v) = (v - 0.5, v + 0.5)$. Thus, assuming that $x_1 = y - 0.5$ and $x_2 = y + 0.5$, $Pr(y)$ is computed by the following formula:

$$Pr(y) = CDF_y(x_2) - CDF_y(x_1) = CDF_{TTF}(p/x_2) - CDF_{TTF}(p/x_1)$$  \hfill (8)

In the case of MTTF, the typical SLA constraint that should be monitored and forecasted is $MTTF \geq K$ (the largest the MTTF the less frequent the failures of the given services) and the probability of violating this constraint can be estimated approximately by the formula:

$$Pr \left( \bigcup_{j=1}^{m} E_j \right) = \begin{cases} 1 - \sum_{j=1}^{m} Pr(y) \times Pr(MTTF_j \geq MTTF_{\text{goal}}), & MTTF < K \\ \sum_{j=1}^{m} Pr(y) \times Pr(MTTF_j < MTTF_{\text{goal}}), & MTTF \geq K \end{cases}$$  \hfill (9)

The above formula is derived similarly to the case of MTTR but due to space restrictions the details of its derivation are omitted (see [24] for a full account).

3 Runtime Computation Of MTTR/MTTF Models

The prediction models of MTTR and MTTF have been implemented using EVEREST+. EVEREST+ is a general-purpose framework for monitoring functional and QoS properties of distributed systems at runtime, and developing prediction models for QoS properties. The framework includes two subsystems: (1) a core monitoring subsystem, called EVEREST (EVEnt REaSONing Toolkit [16]), and (2) a prediction subsystem. The monitoring subsystem of EVEREST+ checks functional and QoS properties based on events intercepted from services using internal or external event captors. Whilst monitoring QoS properties, EVEREST stores QoS related information, including computed QoS property values, instances of violations and satisfaction of guaranteed constraints of QoS properties that have been set in SLAs, and the values of any other state variables that might have been taken into account in checking particular QoS properties. These types of information are made available through an API.

![Figure 3. EVEREST+ Architecture](image)

The prediction subsystem of EVEREST+ (see prediction framework in Figure 3) supports the generation of predictions for potential violations of guaranteed constraints for QoS properties upon request. This support is available through generic functionalities including: the automatic fitting of different built-in probability distribution functions (PDFs) to different types of historical QoS data generated by EVEREST; the selection of the PDFs that have the best fit with the data, the update of PDFs following the accumulation of further QoS property monitoring data; and the generation of predictions for QoS property violations based on built-in and/or user defined functions making use of the probabilities returned by the fitted PDFs.

EVEREST+ automates the prediction generation process based on prediction specifications expressed in an extension of the SLA specification language SLA* [22] that we have developed for this purpose.

3.1 Prediction Specifications

A prediction specification (PS) includes: (a) a set of generic parameters for the forecast, (b) the agreement term to be forecasted, (c) a predictor configuration, and (d) a QoS specification. In the following, we examine these parts of prediction specifications more closely and illustrate them through the example PS shown in Figure 4, which is used to derive the prediction model for MTTR (the grey text within parentheses in the figure shows the differences of the PS for MTTF prediction model).

(i) Generic parameters: The generic parameters in a PS determine: the identifier of the service and the operation that the QoS property to be forecasted relates to, the prediction window of forecasts (i.e., the time period in the future that the prediction is required for), and the history size of forecasts (i.e., the size of the historic event set that will be analysed to derive the prediction model). In the example of Figure 4, the PS refers to the operation Ping of service Srv and sets the prediction window to 10 minutes, and the history size to 500 events.

(ii) Agreement term: The agreement term in a PS specifies the constraint that should be satisfied for the QoS property of interest by the target service and operation of the PS (aka guaranteed state). In the example in Figure 4, the guarantee state refers to the QoS property MTTR (MTTF) and the specified constraint regarding is that MTTR (MTTF) must be less than (greater than) 10 seconds.
(iii) Predictor configuration: The predictor configuration in a PS indicates which prediction model to use for computing the probability of the guarantee state of the PS, and the variables whose probability distribution functions will need to be determined from historical monitoring data as they will be needed by the predictor. The former is specified as the value of the attribute predictor.id of the predictor configuration in a PS and the latter are specified by the element prediction variables that includes a list of variables, each of which is specified by a name/value pair.

```plaintext
1 prediction_specification {
2  generic_parameters {
3    service.id = Srv
4    operation.id = Ping
5    prediction.window.value = 10
6    prediction.window.unit = minute
7    history.window.size = 500
8    history.window.unit = event
9  }
10  agreement_term {
11    expression.qos = MTTR (MTTF)
12    expression.operator = less_than (greater_than)
13    expression.value = 10
14    expression.unit = second
15  }
16  predictor_configuration {
17    predictor.id = MT_SV_PRED
18    prediction_variables {
19      name = EVEREST+.model.distribution
20      value = MTTR (MTTF)
21    }
22  }
23  qos_specification {
24    specification.name = MTTR (MTTF)
25    specification.value = MTTR-Formulas (MTTF-Formulas)
26  }
```

Figure 4. Prediction specification for MTTR/MTTF

In Figure 4, the predictor configurator identifies the predictor as “MT_SV_PRED” (i.e., a parametric predictor for formulas (3) and (9) – see Sect. III.C). It also identifies the two variables used as parameters for the two models (i.e., “MTTR” and “TTF” for the MTTR model and “MTTF” and “TTF” for the MTTF model). The names used for the identification of the two prediction parameters of the MTTR model correspond to names of fluent-variables used in the operational monitoring specifications of the relevant QoS properties. Based on this convention, EVEREST+ can identify the monitoring data that need to be considered in determining the probability distributions functions of the relevant variables at runtime, as we explain below.

(iv) QoS Specification

The QoS specification within a PS provides the operational monitoring specification of the guaranteed state of the QoS property that the prediction is required for. This monitoring specification is expressed in the Event Calculus based monitoring language of EVEREST+, called EC-Assertion. A full description of EC-Assertion is beyond the scope of this paper and can be found in [16]. In the following, however, we provide an overview of the language to enable the reader understand the monitoring specification of MTTR listed below. In EC-Assertion, a guaranteed state over a QoS property is expressed by a monitoring rule and a set of zero or more assumptions. Both monitoring rules and assumptions in have the general form: body⇒head. The semantics of a monitoring rule of this form is that when the body of the rule evaluates to True, its head must also evaluate to True. The semantics of an assumption is that when the body of the assumption evaluates to True, its head can be deduced as a consequence. The body and head of EC-Assertion rules and assumptions are defined in terms of the following Event Calculus predicates:

(a) The predicate Happens(e,t,R(lb,ub)) which denotes that an instantaneous event e occurs at some time t with in the time range R(lb,ub);
(b) The predicate HoldsAt(f,t) which denotes that a state (a.k.a. fluent) f holds at time t;
(c) The predicates Initiates(e,f,t) and Terminates(e,f,t) which denote the initiation and termination of a fluent f by an event e at time t respectively; and
(d) The predicate Initially(f) which denotes that a fluent holds at the start of the operation of a system.

The QoS specification of the MTTR of a service _Srv in EC-Assertion is shown in Table 1. The formulas in Table 1 check whether the MTTR is always below a given threshold K. More specifically, the rule R1 checks for the MTTR condition violations when a call of operation _O in service _Srv is served after a period of service unavailability. The first two conditions in the rule (see Happens predicates) check whether the operation call has been served, i.e., whether the service has produced a response to the call within d time units. The third condition of R1 (cf. predicate HoldsAt(Unavailable(PeriodNumber,_Srv,_STime),t)) checks whether the served operation call happened at a time when the service has been unavailable, and the fourth condition establishes the MTTR value at the time of the call.

The assumption R1.A1 in Table 1 initiates the fluent Unavailable(PeriodNumber+1,_Srv,_STime),t) to represent a period of service unavailability. This fluent is initiated when a service call occurs (i.e., the call represented by the event _id1) without a response to it within d time units, and at the time of the occurrence of the call the service is not already unavailable (i.e., no fluent of the form Unavailable(PeriodNumber, _Srv, _STime) already holds). The assumption R1.A2 terminates the fluent that represents a currently active period of service unavailability (i.e., the fluent Unavailable(PeriodNumber, _Srv, _STime)), when a served service call occurs whilst the service is unavailable. The assumption R1.A3 updates the fluent MTTRnew variable in Table 1 is provided separately, to make the specification easier to read. In EVEREST+, this formula should be specified inside the fluent MTTR(...) of the last Initiates predicate of R1.A3).
A QoS specification must also include a specification of the monitoring pattern that is used to record the values of the prediction variables, which are used in the prediction model of the relevant PS but are not required for the pure monitoring of the property. An example of such a variable is TTF since it is required for predicting the probability of future MTTR values but it is not necessary for monitoring MTTR itself. Hence, the QoS specification of MTTR includes a specification enabling the monitoring of TTF values. Table 2 shows the assumption used to initiate fluents for keeping TTF values (R1.A4).

### 3.2 Computation of MTTR/MTTF prediction models

The MTTR/MTTF prediction models are computed by EVEREST+ dynamically at runtime based on the predicted identifier in their PSs. This predictor is identified as $MT_{SV}$ in Figure 4. $MT_{SV}$ is a parametric predictor realising the following formula:

$$
Pr\left(\bigcup_{y \neq 0} \left\{ \frac{1}{y} \sum_{n=1}^{y} Pr(y) \times Pr(\text{MT}_{SV} \cap \text{MT}_{SV} ) \right\} \right) = MT_{SV} + \frac{(K \times (N + y) - N \times MT_{SV})}{y}
$$

(10) is a parametric form of formulas (3) and (9) for estimating the probability of a QoS constraint of the form $MT^{*} \circ K$ where $MT^{*}$ is the mean time variable of interest, $y$ is a parameter affecting it, $\circ$ is the relational operation used in the constraint (e.g., $<, >, \leq, \geq$), $\neg\circ$ is the negation of this operation, $MT_{c}$ is the value of the mean time variable at the time of the prediction request, and $MT^{*}_{crit}$ is the critical boundary value that is determined by the different values of $y$ ($MT^{*}_{crit} = (K \times (N + y) - N \times MT_{SV})/y$ as discussed earlier).

When a prediction request is received, EVEREST+ determines the PDF of each of the prediction variables in the PS specifications of MTTR and MTTF (i.e., MTTR, TTF for MTTR and MTTF, TTF for MTTF). This is based on computing the parameters of the alternative PDFs in its built-in PDF function set, and then selecting the one that has the best fit with the last N recorded values of each of these prediction variables (N is the value of the $history.window.size$ variable in the prediction specification). The fit of each of the built-in PDFs is measured by the non-parametric Kolmogorov-Smirnov (K-S) goodness-of-fit test [18], and the probability distribution that has the smallest goodness-of-fit (GoF) value according to the test is selected for each PS variable.

EVEREST+ has built-in implementations of 43 continuous PDFs. The set of these functions can be extended, provided that new PDF implementations adhere to the fixed interface required by EVEREST+ for such functions.

### 4 Experimental Results

#### 4.1 Evaluation of precision and recall

To evaluate the precision and recall of the MTTR and MTTF predictor models, we used monitoring data generated from the invocation of the Yahoo WebSearchService [23]. Through a Java client that we developed to invoke this service, we collected a total of 5500 invocation and 5500 response events. The service response time in these invocations varied between 800 and 5251 milliseconds (ms) with an average of 1146 ms. In the experiments, we considered all invocations with a response time of more than 1000 ms as “non served” service calls (“failures”) (see [17] for a similar definition of failures). Based on this filtering, we obtained 1075 “non served” service operation calls and used them to compute MTTR, MTTF, TTR and TTF values.

The total time range of the 5500 invocations was divided in 9 sub-ranges of equal distance and for each of them we computed the MTTR and MTTF values for the end of the sub-range. We also used five different QoS constraints for MTTR and MTTF, based on different K values. The K values were determined by the MTTR and MTTF values at the end time point $t_{i}$ of each of the nine sub-ranges as: $0.75 \times MT_{SV}$, $MT_{SV}^{*}-1$, $MT_{SV}^{*}$, $MT_{SV}^{*}+1$, $1.25 \times MT_{SV}^{*}$. For each K, we generated predictions using combinations of different prediction window sizes (i.e., 1, 10, 60 and 600 seconds) and different history sizes (i.e., 100, 300 and 500 data points). Hence, we carried out 540 predictions for each of MTTR and MTTF. The precision and recall of these predictions were measured by the following formulas:
Precision = \( \frac{TP}{TP + FN} \) 
Recall = \( \frac{TP}{TP + TN} \)

In these formulas, \( TP \) is the number of true (i.e., correct) positive predictions of QoS constraint violations; \( FP \) is the number of false (i.e., incorrect) predictions of QoS constraint violations; \( TN \) is the number of true predictions of QoS constraint satisfaction, and \( FN \) is the number of false predictions of QoS constraint satisfaction. The criteria for classifying a prediction as a TP, FP, TN or FN are summarized in Table 3.

Table 3. Criteria for TP, FP, TN and FN predictions

<table>
<thead>
<tr>
<th>Positive: Probability of QoS constraint violation</th>
<th>Negative: Probability of QoS constraint violation &lt; 0.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>True: QoS constraint violated at: ( t + p )</td>
<td>TP</td>
</tr>
<tr>
<td>False: QoS constraint satisfied at: ( t + p )</td>
<td>FP</td>
</tr>
</tbody>
</table>

We also investigated the effect on precision and recall of: (a) the size of the historic event set (HS) that was used to generate the QoS prediction model, (b) the size of the prediction window (PW), and (c) the K-S goodness of fit measure (GoF) of the probability distribution functions that underpin the prediction model.

Table 4. MTTR and MTTF precision and recall

<table>
<thead>
<tr>
<th>MTTR</th>
<th>MTTF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prediction window (seconds)</td>
<td>Precision</td>
</tr>
<tr>
<td>1</td>
<td>0.96</td>
</tr>
<tr>
<td>10</td>
<td>0.81</td>
</tr>
<tr>
<td>60</td>
<td>0.77</td>
</tr>
<tr>
<td>600</td>
<td>0.47</td>
</tr>
<tr>
<td>History size (events)</td>
<td>100</td>
</tr>
<tr>
<td>300</td>
<td>0.75</td>
</tr>
<tr>
<td>500</td>
<td>0.76</td>
</tr>
<tr>
<td>Goodness of fit</td>
<td>0.09</td>
</tr>
<tr>
<td>0.18</td>
<td>0.77</td>
</tr>
<tr>
<td>0.27</td>
<td>0.76</td>
</tr>
<tr>
<td>0.36</td>
<td>0.73</td>
</tr>
<tr>
<td>Overall</td>
<td>0.75</td>
</tr>
</tbody>
</table>

The recorded recall and precision measures for MTTR and MTTF are shown in Table 4, grouped by the history size, the prediction window and the GoF measure for the best underpinning PDF of MTTR and MTTF. As shown in the table, the overall precision and recall of predictions for all different combinations of HS and PW were 0.75, 0.63 for MTTR and 0.65, 0.7 for MTTF, respectively.

Precision and recall improved significantly for both models when considering short prediction periods, raising to 0.96 and 0.94 for MTTF and 0.9, 0.94 for MTTR when the prediction window was set to 1 sec. This was expected since as the prediction window gets longer, historic data become less indicative of what might happen at the window end.

As shown in Table 4, the precision and recall of the MTTR and MTTF prediction models were not influenced by the history size and the GoF measure. For the history size, this was expected as HS was sufficiently large (1%, 3% and 5% of the total event log). For GoF, the expectation was that a lower GoF would result in higher recall and precision (due to a better fit of the selected PDF to the data) but the experiments did not indicate any effect.

The effect of the prediction window and the history size on the recall and precision of the two models was also tested using a 2-way analysis of variance (ANOVA) for these two factors. ANOVA confirmed that the only statistically significant effect was that of the prediction window on the precision of the MTTR and MTTF predictions. The observed differences in precision for different prediction windows were significant at \( \alpha=0.01 \) (the \( p \)-values for MTTR and MTTF precision where 0 and 0.0019, respectively). ANOVA also confirmed that the history size and the interaction between history size and prediction window had no significant effect in the recall and precision of either model.

4.2 Efficiency

To evaluate the efficiency of the EVEREST+ implementation of the MTTR/MTTF prediction model, we used extended sets of historic events (varying from 50 to 20000 events) and measured the total time that it took to infer the MTTR/MTTF prediction model, i.e., to fit different PDFs to the historic data. Figure 5 shows the time taken by the PDF inference process (i.e., the estimation of the parameters of each PDF from the historic data set) against different sizes of history event sets.

Figure 5. PDF Inference time vs. history event size

As expected, the inference process took longer as the history size became larger. However, increments were linear. Also, since the size of the history event set does not affect the precision and recall of predictions, the MTTR/MTTF models can be efficiently inferred and updated from relatively small historic event sets for which the experiments demonstrated an efficient inference process (e.g., the PDF inference tool 217ms for 500 events).

5 Related Work

Prediction techniques have been developed to generate forecasts for different properties and aspects of software systems. Such techniques may be classified with respect to the property of the software system that a technique aims to predict and the basic algorithmic approach deployed by it (see for a recent survey [13]).
With respect to the prediction property, there have been techniques focusing on prediction of software systems failures [14][8][11], software aging [6], system parameters such as server workloads, CPU loads and network throughput [10][3], or security properties [12]. With respect to the algorithmic approach, different techniques can be characterised as techniques using time series analysis [6][3]; Markov models (e.g. [14][17]); regression models (e.g., weighted regression [4], linear regression [1], auto-regression [5], trend slope estimation [15]); other statistical models (e.g. seasonal Kendall test [7], various mean time prediction techniques [13]); belief based reasoning [12], FSA based prediction [11], machine learning [9] or hybrid techniques (e.g., [8] which uses Markov models and clustering).

Our prediction approach for software services MTTR and MTTF is, to the best of our knowledge, novel both in terms of its algorithmic basis and its focus on prediction of threshold constraints for these availability properties.

6 Conclusions

In this paper, we have presented a prediction model for software service availability based on forecasts of the mean-time-to-repair (MTTR) and mean-time-to-failure (MTTF) for a service. The proposed models realise a black-box where MTTR/MTTF measures are collected at runtime from captured service invocations and responses and are used for the experimental identification of probability distribution functions for variables that affect MTTR/MTTF. The prediction model does not require any behavioural or fault model of the service whose MTTR/MTTF is to be predicted or of the deployment and communication infrastructures used by it. Also the runtime events that are used to infer the prediction models can be collected at the side of either the provider or the consumer of the service.

The MTTR and MTTF prediction models were tested in a series of experiments with positive initial results with regard to the precision and recall of the prediction signals that they generate. The evaluation has also shown the ability to produce and update the prediction models at runtime efficiently.

Currently we are working on developing prediction models for other aggregate QoS properties of software services (e.g., service throughput), without relying on behavioural, compositional or usage service models since such models are scarcely available.

7 Acknowledgments

The work presented in this paper has been supported by the EU Commission; F7 Project SLA@SOI (No. 216556).

8 REFERENCES


[22] K. Kearney, F. Torelli, C. Kotsokalis. SLA*:An abstract syntax for service level agreements (2010). Deliverable, F7 EU project SLA@SOI. To be published.
