ABSTRACT

Modern software systems are increasingly built out of services that are developed, deployed, and operated by independent organizations, which expose them for use by potential clients. Services may be directly invoked by clients. They may also be composed by service integrators, who in turn expose the composite artifact as a new service. Continuous change is typical of this world. Providers may change services and the deployment infrastructure to meet continuously changing requirements and be more competitive. Clients may change their operational profiles. Changes have a severe impact on the quality of services.

In this paper we address the problem of identifying changes concerning the non-functional behavior of software services managed by external organizations, and consequently considered as black-box artifacts. We define the concept of change-point and provide a statistical technique aimed at identifying it, given an execution trace produced by client invocations. Change-point detection is key to reasoning about changes, diagnosing their cause, and suitably reacting to their occurrence.

Categories and Subject Descriptors

C.4 [Computer Systems Organization]: Performance of Systems—Modeling techniques, Performance attributes; D.2.4 [Software Engineering]: Software/Program Verification—Reliability

General Terms

Performance, Reliability

1. INTRODUCTION

1.1 The Context

Software design and development radically changed in the last decade. Software systems were traditionally designed to operate in a completely known and immutable environment.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

FSE-18, November 7–11, 2010, Santa Fe, New Mexico, USA.
Copyright 2010 ACM 978-1-60558-791-2/10/11 ...$10.00.
and control, because of the plethora of different and intertwined phenomena that may occur in the real world. The approach we describe here aims at providing support to reasoning about changes and diagnosing their causes.

1.2 The Problem and Its Motivations

Our contribution focuses on the problem of change-point detection for black-box services. We observe services as black boxes, that is, without accessing their internals, but instead monitoring and analyzing the data that flow through their interface. Service internals are in fact normally inaccessible to clients, and even service providers—who do have access to them—may find it useful to be able to analyze services without accessing the complex internal details of their implementation.

Observations are data streams, collected by probes at the service interface, which describe the behavior of the service concerning a specific quality of interest and from a specific viewpoint. If performance is the quality of interest, observations may be sequences of timestamped invocations with the associated response time. If instead reliability is the quality of interest, they may be sequences of timestamped invocations with an associated result indicating whether the invocation was served successfully or not. Furthermore, observations may express the viewpoint of (and be collected on behalf of) the client, who monitors the calls issued to external services (e.g., see [2] for a monitoring approach). This is called client-side analysis. Alternatively, in server-side analysis, data are collected by service providers, who monitor the QoS of the exported service, which may be used concurrently by several clients.

More precisely, assume we are interested in reliability, and suppose it is measured as a service’s failure rate. Given a set of observations in a time interval, our change-point detection method can detect (1) if a relevant change took place, (2) the point in time when the change occurred, and (3) the new value of the failure rate after the change occurred. For example, consider Figure 1. Suppose that a change-point is detected at the 12th observation of the stream. We can deduce that a relevant change in the system occurred at a time \( t \) which is greater than the timestamp associated with the 11th observation and less than the timestamp associated with the 12th observation.

More precisely, assume we are interested in reliability, and suppose it is measured as a service’s failure rate. Given a set of observations in a time interval, our change-point detection method can detect (1) if a relevant change took place, (2) the point in time when the change occurred, and (3) the new value of the failure rate after the change occurred. For example, consider Figure 1. Suppose that a change-point is detected at the 12th observation of the stream. We can deduce that a relevant change in the system occurred at a time \( t \) which is greater than the timestamp associated with the 11th observation and less than the timestamp associated with the 12th observation.

![Figure 1: Change-point example.](image)

Change-point detection may be a useful tool for the client. If the detected change implies a SLA violation, the exact knowledge of when the change occurred may be used as a proof of evidence in a dispute with the provider. Detected change(s) may also suggest re-binding to a different provider, who may provide a better service [17]. Change-point detection may also be a useful tool for the provider. The provider, in fact, may trace back from the point in time when an unexpected change in quality occurred to the actions that may have caused that change. By reasoning about the change-point, one might point to the installation of a new library component, which substituted a previously installed component and may be responsible for the unexpected change. An important special case is when the client is a service integrator, who offers an added value service to its clients by aggregating existing third-party services. In this case, the detected change-points and (following the previous example) the updated value of reliability of third-party services may be used to compute an updated value of reliability for the composite service, to assess whether this jeopardizes its ability to satisfy the contracts with clients. This kind of analysis may be performed as discussed in [9].

This paper focuses on change-point detection, i.e., on identifying when a potential problem occurred. Understanding its originating cause and planning for reactions which might behave as remedies are outside its scope, and will be the goal of further investigations. Our main contributions are the definition of the concept of change-point, the illustration of its relevance in service-oriented systems, and the development of a statistical framework supporting change-point detection. We also provide an initial assessment of its effectiveness and limitations by numerical simulations.

The remainder of the paper is organized as follows: Section 2 illustrates the case study in which we apply our approach. Section 3 provides a detailed description of the proposed approach. Section 4 illustrates the simulations which validate the proposed approach. Finally, Section 5 discusses related work and Section 6 concludes the paper describing the current limitations of our approach and future work.

2. A RUNNING EXAMPLE

Let us consider a service for image editing, called ImageModder, which provides a set of APIs to manipulate, transform, and save images in different formats. ImageModder represents a real-world case study. It supports a wide range of applications, such as: (1) healthcare applications—which usually require sophisticated image enhancement techniques for diagnosis purposes—or (2) web applications that offer on-line photo editing. The actual implementation of ImageModder resides on a server and interaction with clients is performed through a publicly available Web service or through a downloadable software component in charge of masking the remote interaction with the server. The API provided by ImageModder comprises the primitives listed in Table 1. A complete documentation of the API is beyond the scope of this paper. Due to the lack of space we focus on the description of the essential concepts needed to understand the basic features of ImageModder.

Clients must first authenticate. If authentication succeeds, clients can upload and modify images in an virtual file system organized in folders. Initially, clients must: (1) set their working folder (potentially creating a new one) and (2) set their current image (potentially importing a new one). Afterwards, Imagemodder is ready to work. Each operation invoked by clients directly affects the working folder or the current image. Image transformations are performed through the application of filters. A filter is a transforma-

---

1ImageModder was inspired by the API provided by an existing on-line application: Picnik [24] currently adopted by several clients (e.g., Flickr [10]) to support photo editing of user generated content.
tion that modifies the current image (e.g., histogram equalization, gamma correction, etc.). Finally, users can save or export their modified image. ImageModder offers to clients up-to-date image processing techniques; the library of existing filters is constantly maintained and updated.

Application developers who need image editing features may exploit ImageModder instead of developing in-house equivalent techniques. Relying on a third-party service, however, may cause the problems we described in Section 1. The behavior of a service may change over time, possibly invalidating the assumptions made when that was chosen. We show how change-point detection can be used by a client to identify changes in the non-functional qualities of ImageModder and estimate the new values. The updated estimates may be used by ImageModder clients to decide if the new behavior is compliant with their requirements or if it is necessary to re-bind their system to another service provider or promote an in-house development of these features.

### Table 1: ImageModder API.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>General</td>
<td>login</td>
<td>Authenticates the user and opens a session</td>
<td>Username and Password</td>
</tr>
<tr>
<td></td>
<td>logout</td>
<td>Closes the current session</td>
<td>N/A</td>
</tr>
<tr>
<td>Folder</td>
<td>createFolder</td>
<td>Creates a new folder in the current one</td>
<td>The name of the folder</td>
</tr>
<tr>
<td>Folder</td>
<td>setCurrentFolder</td>
<td>Sets the current folder</td>
<td>The absolute path of the folder</td>
</tr>
<tr>
<td>Folder</td>
<td>renameCurrentFolder</td>
<td>Renames the current folder</td>
<td>The new name of the folder</td>
</tr>
<tr>
<td>Image</td>
<td>import</td>
<td>Loads a file in the current folder</td>
<td>The file to be uploaded</td>
</tr>
<tr>
<td>Image</td>
<td>export</td>
<td>Returns the current image</td>
<td>N/A</td>
</tr>
<tr>
<td>Image</td>
<td>setCurrentImage</td>
<td>Selects the current image</td>
<td>The absolute path of the image</td>
</tr>
<tr>
<td>Image</td>
<td>renameCurrentImage</td>
<td>Renames the current image</td>
<td>The new name of the image</td>
</tr>
<tr>
<td>Image</td>
<td>filter</td>
<td>Applies the filter to the current image</td>
<td>The filter name</td>
</tr>
<tr>
<td>Image</td>
<td>delete</td>
<td>Deletes the current image</td>
<td>N/A</td>
</tr>
<tr>
<td>Image</td>
<td>save</td>
<td>Saves the current image</td>
<td>N/A</td>
</tr>
</tbody>
</table>

and if the following Markov property holds:

\[
P(X_{n+1} = s' | X_n = s, X_1, \ldots, X_{n-1}) = P(X_{n+1} = s' | X_n = s) = m_{s,s'},
\]

where

- \( S \) is a finite set of states: \( S = \{1, \ldots, k\} \);
- \( s_{init} \in S \) is the initial state;
- \( M : S \times S \rightarrow [0, 1] \) is a transition probability matrix; its element \( m_{s,s'} \) represents the probability of passing from state \( s \) to state \( s' \) and \( \sum_{s' \in S} m_{s,s'} = 1 \).

Markov property (1) means that the probability of going from state \( s \) to \( s' \) is independent of the past transitions. In our approach, DTMCs are used to model both probability of failure and discrete distributions of response time. They are specified by the set of states \( S \) and the transition probability matrix \( M \) which contains probabilities that represent, for example, the probability of failure associated with an operation provided by a service. Equally, DTMCs are represented by their diagrammatic notation or by transition probabilities matrices. We will use the former notation for readability purposes and the latter for the formal description of the statistical technique for change-point detection in Section 3.3.

### 3. CHANGE-POINT DETECTION

As we said in Section 1, our approach focuses on identifying changes in non-functional behavior of black-box services given run-time data extracted by running instances of systems that exploit such services. We consider (1) reliability, expressed as probability of failure and (2) performance, expressed as response-time distribution. The proposed approach exploits models to represent the non-functional behavior of the component under analysis. In particular, we adopt Discrete Time Markov Chains (DTMCs). Section 3.1 introduces DTMCs and Section 3.2 describes how these models are used to detect change points of software services.

#### 3.1 Discrete Time Markov Chains

DTMCs are stochastic processes with the Markov property. They are defined as state-transition systems augmented with probabilities. States represent possible configurations of the system. Transitions among states occur at discrete time and have an associated probability. Formally, a sequence of random variables \( X_0, X_1, \ldots \) is a DTMC with tuple \((S, s_{init}, M)\) if:

\[
P(X_0 = s_{init}) = 1
\]

Consider a service for which we wish to perform client-side change-point analysis. Let us first focus on reliability. As a first step, we need to build a DTMC model of the service. The structure of such model is generated by analyzing: (1) the service’s specification (API, interaction protocol) and the associated SLA, (2) expected usage profile of service invocations (i.e., how the client expects or predicts to use the service). Typically, the DTMC comprises one state for every operation provided by the service plus a set of auxiliary states representing potential failure states. Transitions describe the possible sequences of service invocations (and possible failures). They may be annotated with probabilities representing: (1) probabilities of success or failure, derived from the SLA or (2) the usage profile. In the case of performance, for each operation exported by the API, we build an additional DTMC representing the expected discretized
distribution of its response time. Transition probabilities, as in the previous model, can be derived from the SLA. Server-side analysis is similar. The main differences in the latter case may be in the level of detail of the models, which may be more fine-grain, and in the usage profiles, which account for all users accessing the service.

The reliability model derived for ImageModder by analyzing its documentation\(^2\), API, and by predicting a possible usage profile, is shown in Figure 2(a). This model contains: (1) a state for each operation provided by the ImageModder component, (2) several auxiliary states, such as the ready state, which represent the state of the service after the initialization in which the user sets its current image and working directory, and (3) several additional states—highlighted in grey—which represent potential failures associated with specific operations. In this example, we consider only failures associated with operations having an explicit access to repositories (e.g., import, save, etc.). We also consider a failure associated with the login operation to model potential denial of service of the ImageModder component (e.g., in case of too many concurrent requests). The DTMC in Figure 2(a) takes into account the usage profile predicted by the client; for instance, the probability of creating a new folder (i.e., the value labeling the transition from state 0 to state 3) instead of selecting a pre-existing folder after the initial login (i.e., value labeling the transition from state 0 to state 2). Other probabilities are instead extracted from the SLA subscribed by the ImageModder provider, such as the probability of failure associated with the login state. This probability of failure represents the availability of the service, i.e., an information typically declared by on-line service providers.

Performance models can be derived for ImageModder in a similar manner. Figure 2(b) shows the DTMC modeling response time of the save operation (normalized by the size of the saved image). The number of discretizing intervals is equal to the number of states reachable from the initial state. The probability that the response time is in a given time interval is associated with the transition from the initial state to the state representing that interval. Notice that the number of intervals (and hence states) affects the accuracy of the model. However, the approach does not directly depend on the size of the model, but only on the number of transitions under scrutiny.

Our change-point detection method, as discussed later in Section 3.3 in detail, uses execution traces collected by a monitor to figure out whether the actual behavior of a service complies with a given model and, if not, when the change occurred and which are the new values that characterize the model after the change occurred. For example, should the actual performance model of the save operation change during the utilization of the service with respect to the model in Figure 2(b), the change-point detection method would provide: (1) a refined and accurate version of the model representing the service before the change, (2) a different model representing the service after the change, and (3) an estimate for the change-point \(\tau\) in the trace.

The initial models fed to the change-point detection method may reflect our limited and uncertain a-priori knowledge of the system. Both usage profile and probabilities of failure are in fact hard to predict accurately and SLAs may be inaccurate. As we will see, however, our method is robust: it works correctly also in the case where the initial model is inaccurate, since it can derive accurate estimates from actual observations.

3.3 A Bayesian technique for Change-Point Detection

This section illustrates the mathematical background of our method, which exploits a Bayesian statistical technique for change-point detection and involves a Monte Carlo integration method called Gibbs sampling.

We assume that the interaction between a client and the service is captured by an execution trace. An execution trace \(g\) is a sequence of triplets \((r,s,t)\), where \(r\) is the unique identifier of the operation, \(s\) is the time when the invocation is issued, \(t\) is the time when the invocation is completed (if the invocation succeeds), or the special value FAIL (if the invocation fails). From an execution trace, we can derive a reliability trace, used to detect change-points in reliability, and performance traces, used to detect change-points in performance. For simplicity, and space reasons, the way to derive such traces is only informally described hereafter through examples.

A reliability trace is obtained by scanning the execution trace from left to right and mapping it into sequences of paths on the reliability model, each of which represents an interaction with the service. For example, the sequence of paths \((0, 3, 4, 0, 2, 5, 7, 11, 14, 11, 10, 11, 19)\) on the DTMC of Figure 2(a) represents the sequence of two interactions with the service. The former tries to create a folder and then fails. The latter sets the current folder, imports an image, filters, saves, and logs out. Similarly, a performance trace for an operation \(Op\) is built by projecting the execution trace onto a sequence of non-failing \(Op\) invocations. The difference between the response time and the invocation time for each \(Op\) call is then mapped onto the corresponding transition on \(Op\)'s performance model. For instance in the case of the save operation the sequence of paths \((0, 1, 0, 1, 0, 2, 0, 3, 0, 1)\) represents the performance trace projected from an execution trace that contains subsequent calls of the save operation of duration: \((0.1, 0.05, 0.3, 0.7, 0.07)\). Summing up, each trace is a stream of sequences—representing paths in their corresponding DTMC—into which we search for change-points.

From a mathematical viewpoint the change-point detection method works as follows. Given initial distributions for: (1) two random matrices \(A\) and \(B\), which model the service before and after the change, respectively and (2) a random point \(\tau\) in the trace that identifies the change point, our approach generates updated estimates for \(A\), \(B\), and \(\tau\) exploiting the information provided by the trace \(z\).

The change-point detection method requires the user to provide prior knowledge on \(A\), \(B\), and \(\tau\). Matrix \(A\) corresponds to the DTMC that models the initially expected behavior of the service (matrix \(A\) in Figure 2(a) if reliability is our focus). For \(\tau\), we may assume an arbitrary point of the trace. Finally, for \(B\) one can choose arbitrary values or, for example, a “pessimistic” version of \(A\) (e.g., where all failure probabilities are overestimated). As already mentioned,

\(^2\)Due to the lack of space we cannot provide a complete reference documentation of the service. All the crucial information information is reported in Section 2.

\(^3\)The procedure applies identically to performance models (and performance traces) and to reliability models (and reliability traces) since they are both represented in the same mathematical framework.
our method works correctly no matter which initial values we chose for the statistical procedure of change-point analysis. Such values are just initial seeds that do not affect its correctness, but just the size of the trace that must be analyzed to make the correct prediction. We will give evidence of this property later in Section 4.

The method assumes the structure of the model to be immutable and focuses on its parameters, which may change. Thus the cardinalities of A and B are known, equal, and correspond to the structure of the model provided initially by the system designer. In the case of ImageModder, both A and B are 22x22 matrices for the reliability model and 5x5 matrices for the performance model of the save operation (see Figure 2).

Following a Bayesian approach, we consider A, B, τ as random elements characterized by a prior distribution and we proceed to updating them by computing the joint posterior distribution of A, B and τ given data x: P(A, B, τ | x), and the marginal posterior laws: P(A | x), P(B | x), and P(τ | x). In particular, by exploiting the marginal posterior law P(τ | x), we are able to decide, for a given trace x, if a change-point has taken place in the modeled system and when. For example, if the trace length is n and P(τ = n | x) > γ100%, then we are γ–confident that no change-point is present in our trace and the modeled system is still regulated by matrix A. In this setting, τ = n (and τ = 1) mean no change. By computing the mean value of the posterior distribution P(τ | x) we obtain an estimate of the change-point. In particular, by estimating the change-point with the posterior mean we minimize the quadratic Bayesian risk [3].

As for the prior law of A, B, τ, we assume them to be statistically independent. Moreover, we choose independent Dirichlet distributions for each row in A and B and a uniform distribution for τ. Consequently, considering a system modeled with a DTMC with p states (i.e., A and B are composed by p rows and p columns) we have the following prior probability distribution for A:

\[
P(A) = \prod_{i=1}^{p} D(a_i; \alpha_i) \tag{2}
\]

where \(a_i = (a_{i,1}, \ldots, a_{i,p})\) is the \(i^{th}\) row of A and \(D(a_i; \alpha_i)\) is a Dirichlet distribution of parameters \(\alpha_i = (\alpha_{i,1}, \ldots, \alpha_{i,p})\) as follows:

\[
D(a_i; \alpha_i) = \frac{\Gamma(\sum_{j=1}^{p} \alpha_{i,j})}{\prod_{j=1}^{p} \Gamma(\alpha_{i,j})} \prod_{j=1}^{p} a_{i,j}^{\alpha_{i,j} - 1}
\]

For matrix B we have a similar formulation:

\[
P(B) = \prod_{i=1}^{p} D(b_i; \beta_i)
\]

where \(b_i\) is the \(i^{th}\) row of B and \(D(b_i; \beta_i)\) is a Dirichlet distribution of parameters \(\beta_i\). Finally, the prior distribution for \(\tau\) is:

\[
P(\tau) = \frac{1}{n}, \quad \tau = 1, 2, \ldots, n \tag{3}
\]

By an appropriate choice of parameters \(\alpha_i, \beta_i\), Dirichlet distributions capture in a simple and well-established way the prior knowledge and beliefs regarding the structures of transitions matrices A, B. There are no alternative multivariate distributions that are analytically tractable and have well known structural properties as Dirichlet distributions.

The likelihood of data x is:

\[
f(x|A, B, \tau) = \prod_{i=1}^{p} \prod_{j=1}^{p} a_{i,j}^{N_i,j(\tau)} b_{i,j}^{M_i,j(\tau)}, \tag{4}
\]

where \(N_i,j(\tau)\) is the number of transitions in the trace among state \(i\) and state \(j\) until the change-point \(\tau\) and \(M_i,j(\tau)\) is the number of transitions from state \(i\) to \(j\) after \(\tau\). From Bayes’ Theorem we know that:

\[
P(A, B, \tau | x) \propto f(x|A, B, \tau) \times P(A)P(B)P(\tau) \tag{5}
\]

The desired probability \(P(\tau | x)\) can be obtained by integrating (5) with respect to A and B. This can be extremely difficult to evaluate analytically or even numerically. Gibbs sampling\(^4\) comes into play to solve this issue [5, 6, 15, 27], since it allows us to extract samples from \(P(\tau | x)\), without

\(^4\)A description of the Gibbs sampling is beyond the scope of this paper, a brief introduction to this topic can be found in the Appendix.
requiring an explicit computation of it, but exploiting the following marginal conditional distributions:

\[ P(A|B, \tau, z) \quad P(B|A, \tau, z) \quad P(\tau|A, B, z) \]

Let us first compute \( P(A|B, \tau, z) \). By applying Bayes’ Theorem we obtain:

\[ P(A|B, \tau, z) = \frac{P(A, B, \tau|z)}{P(B, \tau|z)} = \frac{P(A)f(z|A, B, \tau)}{\int f(z|A, B, \tau)P(dA)} \] (6)

Using the expression of the likelihood given in (4) and the prior distribution for \( A \) given in (2), for the numerator in (6) we obtain:

\[ P(A)f(z|A, B, \tau) \propto \prod_{i,j=1}^{p} a_{i,j}^{\alpha_{i,j} + N_{i,j}(\tau) - 1} b_{i,j}^{M_{i,j}(\tau)} \]

So that:

\[ P(A|B, \tau, z) \propto \prod_{i,j=1}^{p} a_{i,j}^{N_{i,j}(\tau) + \alpha_{i,j} - 1} \] (7)

We conclude that, conditionally on \( \tau \) and \( z \), \( A \) and \( B \) are independent, i.e. \( P(A|B, \tau, z) = P(A|\tau, z) \), and

\[ P(A|\tau, z) = \prod_{i=1}^{p} D(a_{i}; \alpha_{i} + N_{i}(\tau)), \] (7)

where \( N_{i}(\tau) \) is the total number of transitions from \( i \) until \( \tau \). Reasoning in the same manner for \( B \), we obtain:

\[ P(B|A, \tau, z) = P(B|\tau, z) = \prod_{i=1}^{p} D(b_{i}; \beta_{i} + M_{i}(\tau)), \] (8)

where \( M_{i}(\tau) \) is the number of transitions from \( i \) after \( \tau \).

Concerning instead \( \tau \), it follows from Bayes’ Theorem that:

\[ P(\tau|A, B, z) = \frac{f(z|A, B, \tau)P(\tau)}{\sum_{\tau=1}^{n} f(z|A, B, \tau)P(\tau)} = \frac{f(z|A, B, \tau)}{\sum_{\tau=1}^{n} f(z|A, B, \tau)} \] (9)

where (9) holds if \( P(\tau) \) is the uniform distribution given in (3).

By exploiting Equations (7)-(9) and providing starting values for \( A, B \) and \( \tau \) we can build a Gibbs sequence for each row of \( A \) and \( B \) and for \( \tau \):

\[ a_{k}^{i} \sim D(a_{i}; \alpha_{i} + N_{i,j}(\tau_{k-1})) \]

\[ b_{k}^{i} \sim D(b_{i}; \beta_{i} + M_{i,j}(\tau_{k-1})) \]

\[ \tau_{k} \sim P(\tau|A^{k}, B^{k}, z) \]

where \( a_{k}^{i}, b_{k}^{i}, \) and \( \tau_{k} \) correspond to the \( k^{th} \) sample of the Gibbs sequence. Iterating this sampling process \( N \) times, we obtain a sequence: \( (\tau_{1}, a_{1}^{1}, b_{1}^{1}, \ldots, \tau_{N}, a_{N}^{N}, b_{N}^{N}) \) that converges to \( P(A, B, \tau|z) \). In particular for \( N \) large enough the last values \( \tau_{m+1}, \ldots, \tau_{N} \) where \( m < N \), can be considered as \( N - m \) samples from \( P(\tau|z) \) and can be used to estimate \( \tau \). In the same manner, the sequence \( (a_{1}^{m+1}, \ldots, a_{N}^{m}) \) can be considered as \( N - m \) samples from \( P(A|z) \) and \( (b_{1}^{m+1}, \ldots, b_{N}^{m}) \) as \( N - m \) samples from \( P(B|z) \) used to estimate \( A \) and \( B \), respectively.

Table 2: Performance: Discrete Distribution of Response Time (RT).

<table>
<thead>
<tr>
<th>RT</th>
<th>Before change-point</th>
<th>After change-point</th>
</tr>
</thead>
<tbody>
<tr>
<td>( RT &lt; 0.2 )</td>
<td>0.1</td>
<td>0.4</td>
</tr>
<tr>
<td>0.2 &lt; ( RT &lt; 0.5 )</td>
<td>0.4</td>
<td>0.1</td>
</tr>
<tr>
<td>0.5 &lt; ( RT &lt; 0.8 )</td>
<td>0.4</td>
<td>0.1</td>
</tr>
<tr>
<td>( RT &gt; 0.8 )</td>
<td>0.1</td>
<td>0.4</td>
</tr>
</tbody>
</table>

It is important to notice that we conceived an approach based on the Bayesian estimation theory since simpler approaches cannot achieve the same level of precision. For example, approaches based on rolling averages computed over the streams of data are dependent on the size of the window used to compute the average, which is dependent on the variance of the data (an unknown parameter in our domain). Moreover, simpler approaches based on computing and comparing \( P(A|z) \) against \( P(B|z) \) are less precise since they do not provide any estimate for \( \tau \).

4. VALIDATION

A Java implementation of the change-point detection method presented in this paper has been publicly released\(^5\). The method has been validated by simulation using the ImageModder case study. Precisely, we developed a client application that invokes the ImageModder service, we collected execution traces, and we analyzed them. The simulated service can be instructed to change its behavior, by setting change-points that affect its performance and/or reliability. In the real world, the changes may be consequences of software updates. In this section, we describe a significant subset of the simulations we performed to validate the approach.

Due to the lack of space, we report on a limited number of cases and restrict our discussion to only one quality attribute, namely performance of the save operation, which is modeled by the DTMC in Figure 2(b). In the experiments we describe below, we changed the server’s response time distribution after specific invocations and thus simulating change-points. Afterwards, we ran a program that implements our change-point analysis approach in different scenarios and with different settings as explained hereafter. The results we report hold for traces of 400 invocations to the save operation tracing the response time, normalized by the size of the saved image. For each case we discuss below, we ran 1000 simulations. The findings obtained by focusing on performance of this operation apply identically to any other operation and to the reliability model. The interested reader may repeat our experiments and perform others by using our downloadable tool.

**Single Change-point Detection.** We first report on simulations which consider a single change-point occurring in the observation period. As for the Gibbs sampling, we used the Single Sequence approach and a sequence of length 1000 with a burn-in of length 700. As for prior parameters \( \alpha_{i} \) and \( \beta_{i} \) of the Dirichlet distribution of \( A \) and \( B \) we used, respectively, the DTMC in Figure 2(b) and a DTMC with equally distributed probabilities attached to all outgoing transitions from state 0.

\(^5\)http://home.dei.polimi.it/tamburrelli/ChangePoint/
Table 3: Characteristics of Posterior Distribution.

<table>
<thead>
<tr>
<th>Gibbs Sequence Length</th>
<th>Min</th>
<th>Median</th>
<th>Mean</th>
<th>Max</th>
<th>Average Error</th>
<th>Max Error</th>
<th>$P(140 \leq \tau \leq 160)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>141</td>
<td>151</td>
<td>151.8227</td>
<td>162</td>
<td>0.01215141</td>
<td>0.01316296</td>
<td>0.999967</td>
</tr>
<tr>
<td>1000</td>
<td>140</td>
<td>151</td>
<td>151.8181</td>
<td>165</td>
<td>0.01212084</td>
<td>0.01540741</td>
<td>0.9999689</td>
</tr>
</tbody>
</table>

Figure 3: Average Posterior Distributions of $\tau$ with change-point at 150.

The latter two choices represent the worst possible scenario in which we have no clue about where the change-point will potentially take place and which will be the new performance model. Finally, we changed the server’s response time distribution in the 150th invocation as shown in Table 2 (i.e., the 150th invocation represents the change-point). By running our change-point analysis method in the aforementioned scenario we obtained the average posterior distribution for $\tau$ illustrated in Figure 3 over 1000 simulations. The posterior distribution of $\tau$ is concentrated around 150, as expected. More precisely, it presents the characteristics listed in row 2 of Table 3. By synthesizing an estimation for $\tau$ with the mean value of the posterior distribution, we obtain an average estimate equal to 151.8181. By computing the posterior probability $P(140 \leq \tau \leq 160)$, as shown in last column of Table 3, we are at least 99% confident that a change-point took place between 140 and 160.

Gibbs Sequence Length. We now report on simulations concerning the length of the Gibbs sequence. Intuitively, the longer the sequences are the more precise our distribution and our estimate will be. As shown in Table 3, sequences of length equal to 1000 give a very accurate result. However, we performed the same testbed with shorter sequences to stress the robustness of the approach (we decreased the number of samples down to 100). In this extreme case, illustrated in row 1 of Table 3, the mean value (i.e., the change-point estimate) remains close to 150. Generally speaking, the accuracy of estimates begins to decay using sequences shorter then 400, as shown in Figure 4, where the estimation error ($E$) is computed as:

$$E = \frac{|ec - rc|}{rc}$$

$ec$ being the estimated change-point and $rc$ being the real change-point. Notice that the average estimation error with sequences longer then 400 is always less then 0.01214. From Table 3, we can conclude that our approach is quite robust, since the average and maximum estimation errors are almost negligible. Finally, the appropriate Gibbs sequence length can be automatically determined by adopting the Potential Scale Reduction Factor as described in [16].

Estimates of $A$ and $B$. The considerations made for the estimate of $\tau$ holds also for the estimates of $A$ and $B$. Figure 5(a) shows the histogram of the transition from state 0 to state 1, with response time $< 0.2$, over 1000 simulations. It corresponds to the mean value of the posterior distribution computed exploiting samples in the Gibbs sequence. Notice how the average estimate is concentrated around the real value which generated data (i.e., 0.1). Figure 5(b) shows the same data obtained for the transition from state 0 to state 1 in matrix $B$, which is concentrated around the desired value 0.4. By adopting again the mean value of the posterior distribution to estimate transition probabilities, we obtain

Figure 4: Average Estimation Error.

Figure 5: Average Posterior Distribution for Transition 0-1.
an average value of 0.096 for transition 0 – 1 before the change point, with respect to the real value of 0.1, which corresponds to an average estimation error equal to 0.04.

**Change-point Location.** We made experiments to assess the robustness of the change-point detection method with respect to the location of the occurrence of the change-point in the trace. Figure 6 shows the posterior distribution of \( \tau \) when the change described in Table 2 occurs at the 5th invocation of the `save` operation. The distribution is correctly centered around the expected value of 5 and thus it is possible to correctly estimate the presence of a change-point in this position by adopting again the mean value of the posterior distribution.

**No Change-point.** Let us now consider the scenario in which no change-point occurs in the trace; i.e., the service constantly behaves as described by the second column in Table 2. In this setting, by running the change-point detection we obtained the posterior distribution illustrated in Figure 7. As we said in Section 3.3, \( \tau = 1 \) or \( \tau = n \) indicate that no change occurred in the trace. Figure 7 shows that the posterior distribution is centered around the beginning of the trace, which approximates \( \tau = 1 \) and thus indicates no change.

**Sensitivity to the Initial Model Values.** In the previous experiments, the initial values for matrix \( B \) a DTMC are equally distributed probabilities attached to all outgoing transitions from state 0, which represent the worst possible scenario in which we have no clue about which will be the new performance model. Figure 5(b) shows that the change-point detection algorithm produces precise estimates of transitions in matrix \( B \) even though such initial values reflect inaccurate knowledge. The initial value of matrix \( A \) also does not affect the robustness of the method. In fact, we first ran the detection method by initializing matrix \( A \) with the data of columns 2 in Table 2, which correspond to the actual initial values of performance of the `save` operation (a “perfect knowledge” situation), and then we ran the method with opposite initial values, i.e. \( (0.4, 0.1, 0.1, 0.4) \). Figure 8 shows the convergence of the Gibbs sequence which estimates transition 0 – 1 of matrix \( A \). The figure shows that the convergence proceeds similarly in both cases, except for initial fluctuations in the case of inaccurate initial values.

**Multiple Change-points.** We performed simulations to check the behavior of the method if multiple change-points occur in the observation period. We injected a change-point at invocations 150 and 250, over the 400 invocations. Figure 9 shows that as a result of simulations, the posterior distribution of \( \tau \) is clearly a bi-modal distribution: each peak in the distribution indicates a change-point. Once these peaks are identified, it is possible to divide the trace in two distinct traces which separate the peaks and, by re-running the change-point detection, we can reduce the problem to the case of a single change-point.

**Performance and Multiple Sequence Gibbs Sampling.** The proposed approach is quite efficient. The computation of a single Gibbs sequence of length 1000 with a burn-in of 700 requires about 3.5 seconds to analyze a trace of 400 invocations on a conventional workstation\(^6\). Execution time can be reduced by examining shorter sequences and by applying the Multiple Sequence variant. It is possible to show (but

\(^6\)Intel®Core 2 Duo with 4Gb RAM. Implementation in Java v1.6.
space reasons do not permit it) that the variant produces estimates of equal accuracy, but improves time efficiency. For example, the same workstation can execute the parallel computation of two Gibbs sequences of length 500 with a burn-in of 300 in about 1.7 second to analyze the same trace of length 400.

5. RELATED WORK

Change-points are abrupt variations in the generative parameters of a data stream. Their identification has found important application in several disciplines, such as finance, biometrics, and robotics (e.g., see [20] or [28]). In particular, many works in the area of intrusion detection systems aim at detecting when a change/violation occurs given a trace log and exploiting Bayesian techniques as described for example in [4]. To the best of our knowledge, no existing work applied these concepts to software reliability and performance, and in particular to SOC.

DTMCs and other stochastic models are increasingly used to assess dependability of software artifacts (e.g., see [19]) and to predict service performance and reliability (e.g., see [25]). The problem of dealing with changes in the external services used by a composite service, and adapting the parameters of its quality model accordingly, is studied in [9, 18]. A framework for component reliability prediction in presented in [8], whose objective is to construct and solve a stochastic reliability model, through which software architects may explore competing designs. In particular, the authors tackle the definition of reliability models at architectural level and the problems related to parameter estimation. Other complementary approaches investigate alternative methods for calibrating model parameters at run time in the context of performance models [29].

The problem of quality of service in SOC is studied by several authors, who focus on how quality can be specified and how it can be the basis for verifiable SLAs. A language for SLAs has been proposed by [26]. Other related areas deal with monitoring and verifying services and service compositions [2, 13, 14]. Run-time verification is another closely related research area (for example, see [7]). Several approaches have been proposed in literature that deal with non-functional aspects of services and their composition. In particular, [22] illustrates a framework for modeling and evaluating service-oriented applications and [21] describes performance prediction in the SOC domain exploiting Queuing Networks.

An approach for verifying service compositions starting from UML descriptions and then transforming them into a specific representation that allows validation with respect to concurrency properties is presented in [12]. A similar approach is described in [11], which shows how to verify service integrations in case of resource constraints, with respect to safety and liveness properties.

Concerning the statistical techniques we adopted for change-point detection, Carlin et al. in [5] provide a hierarchical Bayesian analysis of change-point problems that inspired our work and suggested the adoption of Gibbs sampling. In particular, concerning this integration method, Casella et al. in [6] provide a complete discussion about its properties by examples.

6. CONCLUSION AND FUTURE WORK

In this paper we addressed the problem of identifying changes concerning the non-functional behavior of software services managed by third-party entities and considered as black-box artifacts. We defined the concept of change-point and provided a statistical technique aimed at identifying them given an execution trace extracted by running instances of the system. Change-point detection was performed concerning reliability and performance through the adoption of DTMCs. We implemented a tool supporting change-point analysis as part of the KAMI framework—a toolset which is illustrated in [9, 18]. The tool has been used to validate the method via simulations. We performed extended simulations, but for space reasons we could only report on selected cases. For instance, we omitted some interesting results concerning the relation between the length of the trace and the range of values of probabilities appearing in the models, and the relation between the length of the trace and the distance between different change-points (in a multiple change-point setting).

In the future, we plan to complement the simulation-based validation with analysis of existing on-line services, to obtain quantitative result in a real-world setting. In addition, we will investigate how and when change-point detection can be run to support on-line reactions to detected changes and, more generally, we will explore the trade-offs between on-line and off-line change detection and their dependence on the temporal behavior of the application. Further work may also apply change-point analysis to other models such as Queuing Networks or continuous Markov chains.

Appendix: Gibbs Sampling

Gibbs sampling is an integration method aimed at computing characteristics (such as the mean or variance) of the marginal density \( f(x) \) of a joint density \( f(x, y_1, \ldots, y_m) \) without requiring to actually compute the integral \( f(x) = \int \ldots \int f(x, y_1, \ldots, y_m) \, dy_1 \, dy_2 \ldots dy_m \), which can be extremely difficult to perform analytically or even numerically. In particular, Gibbs sampling allows for generating a sample \( X_1, \ldots, X_n \) from \( f(x) \), without calculating \( f(x) \). This is because Gibbs sampling works with the (univariate) conditional distributions of every random variable \( X, Y_1, \ldots, Y_m \) given all the other ones: \( f(x|y_1, \ldots, y_m), f(y_1|x, y_2, \ldots, y_m), \ldots, f(y_m|x, y_1, \ldots, y_{m-1}) \). To briefly illustrate how Gibbs sampling works, let us consider the simple two-variables case in which we extract samples from the marginal distribution \( f(x) \) of a joint density \( f(x,y) \), by sampling from the univariate conditional densities \( f(x|y) \) and \( f(y|x) \).

The sampler starts with some initial value \( y_0 \) and generates \( x_0 \) by sampling from \( f(x|y = y_0) \). Thus the sampler uses \( x_0 \) to generate a new value \( y_1 \), drawing from \( f(y|x = x_0) \). Hence,
the sampler proceeds as follows:

\[ x_i \sim f(x|y = y_{i-1}) \]
\[ y_i \sim f(y|x = x_i) \]

By iterating this sampling process \( k \) times, we obtain a Gibbs sequence of length \( k \): \((x_1, y_1), \ldots, (x_k, y_k)\). If we think of each \((x_i, y_i)\) as a realization of a random vector \((X_i, Y_i)\), then, under mild conditions, as \( k \to \infty \) the distribution of \((X_k, Y_k)\) converges to the joint density \( f(x, y) \) (independently of the starting value \( y_0 \)) and hence the distribution of \( X_k \) converges to marginal distribution \( f(x) \) (see for example [27]). Hence, for large enough \( k \), the last values \( x_{h+1}, \ldots, x_k \) \((h < k)\) of the Gibbs sequence can be considered as \( k - h \) samples from \( f(x) \). It is important to repeat the sampling process a sufficient number of times to have a large Gibbs sequence and to ignore (as shown by Figure 10(a)) the initial samples (burn-in removal) which are not distributed according to \( f(x, y) \) and are influenced by the starting values. Alternatively, instead of collecting samples from a large enough Gibbs sequence, Gelfand et al. in [15] suggest the generation of \( m \) independent Gibbs sequences of length \( k \) and then using the \( m \) final values of these sequences, as shown by Figure 10(b). The choice of \( k \) and alternative approaches to extracting information from the Gibbs sequence, are discussed in [6]. A complete description of the Gibbs sampling is beyond the scope of this paper, further details can be found in [5, 6, 15, 27]. We explicitly briefly introduced and justified it because it is a crucial component of our solution.

Acknowledgments
This research has been partially funded by the European Commission, Programme IDEAS-ERC, Project 227977-SMSCom.

7. REFERENCES