Analyzing Web Logs to Detect User-Visible Failures

Wanchun Li
Georgia Institute of Technology

Ian Gorton
Pacific Northwest National Laboratory

Abstract

Web applications suffer from poor reliability. Practitioners commonly rely on fast failure detection to recover their applications quickly to reduce the effects of the failures on other users. In this paper, we present a technique for detecting user-visible failures by analyzing Web logs. Our technique applies a first-order Markov model to infer anomalous browsing behavior discovered in Web logs as indicators that users have encountered failures. We implemented our technique in a tool called REBA (REcursive Byesian Analysis of Web Logs). We evaluated our technique using REBA applied to the Web site of NASA. The results demonstrate that our technique can detect user-visible failures with reasonable cost.

1 Introduction

Web applications suffer from poor reliability. Practitioners commonly rely on fast detection of user-visible failures (i.e., failures encountered by users but not detected by in-house testing), and expect to recover their applications quickly to reduce the effects of the failures on other users. Research shows that early detection could mitigate about 65% of failures [13].

However, fast failure detection in Web applications is challenging—it can require as much as 75% of failure recovery time [5]. One possible way for practitioners to quickly detect failures is to seek users' feedback. However, most users are reluctant to give feedback. Moreover, even if users do provide feedback, it may not be useful for detecting failures. Our industrial collaborator, Clinical Guard, an online medical device supplier, reported that in three months of 2008, their application operators sent enquiries to 200 customers whose browsing histories showed anomalous browsing paths, which operators believed might be generated because these customers encountered failures. The operators received only nine responses. Of these, only one identified the failure and provided information that was helpful in detecting the failure. The remaining eight users just mentioned that they encountered failures when they tried to buy products through the Web site.

Previous research has addressed failure detection in Web applications. One approach concentrates on performance diagnosis by detecting failures of network services [2, 7]. Many failures identified by these techniques downgrade the performance but do not affect the functionality of a Web application. Another approach investigates detecting failures at the application level [6, 11]. These techniques trace runtime execution paths that consist of components invoked in response to user's requests, and applies statistical analysis to infer whether failures occurred with respect to anomalous interactions of these components. The techniques can successfully detect application level failures, but may not be effective for detecting failures caused by software bugs [11]. A third approach analyzes users' browsing behavior to detect user-visible failures [3]. The technique reported in Reference [3] assumes that users refresh their browsers when they encounter failures. The technique detects abrupt changes in the hit rates of a Web page and notifies human operators to perform further checking. The technique can detect failures caused by software bugs for which users respond with refreshing browsers. However, the technique could miss failures if the number of refreshing responded for failures is not enough to reach the threshold of abrupt (e.g., users respond with other behavior than refreshing for failures).

To address the limitations of these existing techniques, we developed a technique for detecting user-visible failures in Web applications. Our technique assumes that the HCI rational principle holds (i.e., users must respond

if the output of a sequence of interactions with software is unsatisfactory) [4]. Web users normally browse Web pages following certain browsing patterns [10, 12, 14]. Users' reaction to failures may break these browsing patterns and thus, has a low likelihood of occurrence. Our technique applies a first-order Markov model to model users' browsing behavior, uses the data in Web logs to train the Markov model, and applies the trained model to infer unusual browsing paths (i.e., sequences of requests from users with low probability of occurrence). If an unusual browsing path is detected, our technique reports that the user has encountered a failure.

We implemented our technique in a tool called REBA (REcursive Byesian Analysis of Web Logs), and evaluated the technique using REBA applied to NASA's Web site. We built the Markov model using user-session data retrieved from a one-month Web log. We used Bayesian inference to estimate the parameters of the model: the user-session data from the first day were used in setting the prior distribution to estimate the parameters, the user-session data from the second to the tenth days were used as training data to estimate the parameters, and the user-session data for the remaining days were used to test the model. Our study shows that our technique detects approximately 71% of the failures with a 26% false-positive rate. We analyzed the results using the measures of ROC (Receiver Operating Characteristic) curve [9], a broadly-used graphical plot for evaluating and visualizing the performance of statistical models, and the analysis indicated that our model can detect failures with reasonable cost.

The main contributions of this paper are

- A new technique for detecting user-visible failures in Web applications that builds a probabilistic model of user-browsing behavior
- A prototype tool, REBA, that implements our technique
- The results of an empirical study on a real-world application that demonstrates the effectiveness of our technique

2 Overview of the Technique

In this section, we discuss the assumptions of our technique (Section 2.1), present our model (Section 2.2), describe the way of using the model to detect failures (Sections 2.3), and discuss important features of our technique (Section 2.4).

2.1 Assumptions

Our technique assumes that HCI rational principle holds that a user will respond when the output of a sequence of interactions with the software is not satisfactory [4]. Research has shown that users follow browsing patterns to use a Web site [10, 12, 14]. We assume that when users encounter failures, their reactions to the failures often break browsing patterns and the probability of the occurrence of such response is low. Thus, if the occurrence probability of several successive requests is low, the user making these requests may have encountered failures.

2.2 The Model

A Markov model describes a system consisting of a set of N distinct states [15]. The system transits from the *source state* to the *sink state*. A Markov model can be represented by a directed graph whose nodes represent states and edges represent transitions (i.e., the start node of an edge represents the source state and the end node represents the sink state). A Markov model transits from a source state to a sink state according to a conditional probability, which is called the *transition probability*. If the transition probability to the next state is conditionally dependent on only the current state, the model is at the first-order.

Our technique builds a first-order Markov model to analyze user browsing behavior. The technique employs user-session data retrieved from Web logs to train this model, and then uses the model to compute the occurrence probabilities of sequences of user requests. If a sequence's probability is lower than a threshold value, the sequence is reported as one in which a user encountered a failure.

In our Markov model, a state represents a Web page, and a transition represents navigation from one page to another. The direction of a transition represents the direction of the navigation. For example, if a user first visits page A and then visits page B, the corresponding transition in our model is from the state representing A to the state representing B. If a user refreshes page A, this refreshing operation corresponds to the self-loop transition from A's state back to itself.

States and transitions of our model have associated *occurrence numbers*. A state s has a *state occurrence number*, denoted by $Occ(s)$, which is the number of times that s was visited; a transition $s_i s_j$ has a *transition occurrence number*, denoted by $Occ(s_i s_j)$, which is the number of times that the system made the transition $s_i s_j$.

2.3 Failure Detection

Our technique uses a trained Markov model to estimate transition probabilities of a sequence of transitions to infer anomalous browsing paths, which are viewed as indicators that the user has encountered failures.

In our model, there are two types of transition probabilities: outgoing and incoming. Given a transition $s_i s_j$, the *outgoing transition probability* (OTP) is the probability that from state s_i the system will transit to state s_j; the *incoming transition probability* (ITP) is the probability that when the system is in s_j, the system transited to s_j directly from s_i. For example, given the Home page H and another Web page A, the OTP of the transition from H to A is the probability that a user navigates from the Home page to page A; while the ITP is the probability that when a user is browsing page A, the user navigated to the page directly from the home page. OPT and ITP are usually different. For example, users may go from any Web page to the Home page, but not vise versa.

To check whether a user encountered a failure in a specific request, our technique checks the *outgoing occurrence probability* and the *incoming occurrence probability* of a fixed number $1 + w$ of subsequent requests (i.e., this request and its w successive requests). Given a sequence T of requests and a trained Markov model M, the *outgoing occurrence probability* (OOP) is the occurrence probability of T computed using OTPs, whereas the *incoming occurrence probability* (IOP) is computed using ITPs. If the minimum of OOP and IOP of the subsequence of requests is less than a threshold, our technique views T as an anomalous browsing path and reports that the user encountered failures. For example, given a user session, if we want to check whether a user encountered failures in the fourth request, the model estimates the OOP and the IOP of the sequence of transitions from the 4th request to the $(4 + w)$th request, where w is a predefined value. If the minimum of OOP and IOP of this sequence of transitions is less than a specific threshold, the technique reports that the user encountered failures in the 4th request.

2.4 Discussion

We discuss three important features of our technique.

Choice of the model. A first-order Markov model may not perfectly model a user's browsing behavior, because the latter may not satisfy the first order Markov property. The next page a user visits may depend not only on the current page but also on earlier pages. Although the dependence could be captured by a higher order Markov model, it would entail additional complexity and computational cost. Our empirical results (see Section 4) suggest that a first-order model is adequate for revealing many failures in Web pages.

Computing OOP and IOP. We use Bayesian estimation to compute OTP and ITP. We set the prior to compute the estimator. The way of setting prior ensures that OOP and IOP of a sequence of requests are not dependent on a specific page. For example, given two pages that have 1000 links and 1 link, respectively, the probabilities of visiting the page with more links are not necessarily lower than the page has less links. Rather, the probabilities are dependent on the number of links and the frequencies of these links are visited (See Section 3.3 for more details).

The threshold value. Our model raises an alarm when the occurrence probability of a sequence of requests is lower than a threshold value. The specific value of the threshold depends on applications and the loss functions. Suppose the practitioner chooses the same loss function for detection rate and false positive rate, the threshold value can be determined by the output of the model training.

Limitations of our technique. Our technique does not handle two types of user-browsing behavior. First, users sometimes immediately leave a Web site when they find a failure in their first request. In this case, it is difficult for our technique to distinguish whether the user left the site due to a failure. Second, a user may not take any unusual actions after encountering a failure because the user did not notice or care about it. In this case also, our technique cannot detect the failure.

3 Model Training

Our technique employs Bayesian inference to estimate parameters of the Markov model (i.e., outgoing transition probabilities and incoming transition probabilities). In this section, we introduce notations (Section 3.1), present estimating transition probabilities (Section 3.2), discuss setting parameters of the prior distribution (Section 3.3), and describe updating the model (Section 3.4). Without loss of generality, we consider in detail only estimation of outgoing transition probabilities. The treatment of incoming transition probabilities is similar and is omitted.

3.1 Notations

Let $D = (u_1, ..., u_n)$ be a set of user sessions. For a user session u_q, let $k_i^q = (k_{i1}^q, ..., k_{im}^q)$ be the occurrence numbers of transitions leaving from s_i to m states,

Table 1: Example training user-session data.

User Session	Requests
u_1	$\{s_1, s_2, s_1, s_3, s_1, s_2, s_1, s_1, s_2\}$
u_2	$\{s_1, s_3, s_2, s_1, s_2, s_1, s_2\}$
u_3	$\{s_1, s_2, s_1, s_4\}$

including itself. Let $\theta_i = (\theta_{i1}, ..., \theta_{im})$ be the forward transition probabilities for transitions leaving from state s_i. Let n_{ij} be the total occurrence number of transition $s_i s_j$ in all the user sessions of data set D. We have $n_{ij} = \sum_{q=1}^{n} k_{ij}^q$.

Table 1 shows a simple example of a set of user sessions consisting of three user sessions, u_1, u_2, and u_3. The three user sessions are constructed by four states, s_1, s_2, s_3, and s_4. For user session u_1, we have $\theta_1 = (\theta_{11}, \theta_{12}, \theta_{13}, \theta_{14})$ that are transition probabilities from s_1 to each of the four states, and $k_1^1 = (k_{11}^1, k_{12}^1, k_{13}^1, k_{14}^1)$, where $k_{11}^1 = 1$, $k_{12}^1 = 3$, $k_{13}^1 = 2$, and $k_{14}^1 = 0$, that are the occurrence numbers of transition $s_1 s_1$, $s_1 s_2$, $s_1 s_3$, and $s_1 s_4$.

3.2 Estimating Transition Probabilities

Our technique uses Bayesian inference to learn a first-order Markov model. Bayesian inference views parameters of a model as random variables and infers the distributions of these random variables by following the process of Bayes Theorem. To estimate a parameter, a common Bayesian method is to (1) assume a distribution of the parameter, (2) set a *conjugate prior distribution* (i.e., a distribution in the same distribution family as the assumed distribution), (3) update the conjugate prior using the training data, where the updated prior is the *posterior distribution*, and (4) estimate the parameter using the posterior distribution. One common estimator used in Beyesian estimation is the *posterior mean*, which is the mean of the estimated posterior distribution.

Given a Markov model with m states, according to Bayes' Theorem, we have

$$P[\theta_i|D] = \frac{P[D|\theta_i]P[\theta_i]}{P[D]} \tag{1}$$

where $P[\theta_i|D]$ is the posterior, $P[D|\theta_i]$ is the likelihood, and $P[\theta_i]$ is the prior; $P[D]$ is the probability that the data D is observed, which is a normalization constant in the computation of the posterior.

Equation 1 presents a Multinomial distribution. To estimate the parameters of this Multinomial distribution, we use a Dirichlet distribution with parameters α and $\mathbf{q} = (q_1, ..., q_m)$ as its conjugate prior, where q_j is the prior estimate of the forward transition probability of $s_i s_j$ after s_i is visited α times. The distribution function of a random vector $\theta = (\theta_1, ..., \theta_m)$ governed by a Dirichlet distribution with parameters α and $\mathbf{q} = (q_1, ..., q_m)$ is

$$Dir_{\alpha,\mathbf{q}}(\theta) = \frac{1}{Z}\prod_{i=1}^{m}\theta_i^{\alpha q_i - 1} \tag{2}$$

where θ_i, q_i, $\alpha \geq 0$, $\sum\theta_i = 1$, $\sum q_i = 1$, and $Z = \frac{\prod_{i=1}^{m}\Gamma(\alpha q_i)}{\Gamma(\alpha)}$ is a constant normalization factor.

Thus, according to Equation 2, we have the prior

$$P[\theta_i] = \frac{1}{Z}\prod_{j=1}^{m}\theta_{ij}^{\alpha q_j - 1} \tag{3}$$

Now, given the probability structure $\theta_i = (\theta_{i1}, ..., \theta_{im})$, we assume the independence of the training user sessions. This assumption is reasonable because two user sessions record two different browsing histories, which, particularly generated by two different users, are independent. Thus, we have the likelihood of selecting the data set D as

$$\begin{aligned} P[D|\theta_i] &= P[(u_1, ..., u_n)|\theta_i] \\ &= P[u_1|\theta_i]...P[u_n|\theta_i] \end{aligned} \tag{4}$$

We also have the likelihood of collecting user session u_q, in which transition $s_i s_j$ occurred k_q^{ij} times, regarding to θ_{ij}. As shown in Equation 5, the likelihood is the marginal probability of random variable θ_{ij}, which views other variables in θ_i as 0.

$$P[u_q|\theta_i] = \theta_{ij}^{k_{ij}^q} \tag{5}$$

Substitute Equation 4 using Equation 5, we have

$$\begin{aligned} P[D|\theta_i] = P[u_1|\theta_i]...P[u_n|\theta_i] &= \prod_{j=1}^{m}\prod_{q=1}^{n}\theta_{ij}^{k_{ij}^q} \\ &= \prod_{j=1}^{m}\theta_{ij}^{\sum_{q=1}^{n}k_{ij}^q} = \prod_{j=1}^{m}\theta_{ij}^{n_{ij}} \end{aligned} \tag{6}$$

Then substitute Equation 1 using Equations 3 and 6, we have

$$\begin{aligned} P[\theta_i|D] &= \frac{\prod_{j=1}^{m}\theta_{ij}^{n_{ij}} \cdot \frac{1}{Z}\prod_{j=1}^{m}\theta_{ij}^{\alpha q_j - 1}}{P[D]} \\ &= \frac{1}{Z \cdot P[D]}\prod_{j=1}^{m}\theta_{ij}^{(n_{ij} + \alpha q_j - 1)} \end{aligned} \tag{7}$$

Now, we have Equation 7 as the posterior distribution. We use the expectation of this distribution (i.e., the mean posterior) to estimate transition probabilities of the outgoing transitions of state s_i.

To easily compute the mean posterior, we introduce new symbols to construct a concise form of Equation 7. Because both Z and $P[D]$ are constant normalization factors, as shown in Equations 1 and 3, we can set a new constant $Z' = Z \cdot P[D]$. We also introduce a new parameter $\alpha' = n_i + \alpha$ and a new vector parameter $q'_j = \frac{n_{ij}+\alpha q_j}{n_i+\alpha}$. Substitute Equation 7 with Z', α', and q', we have

$$P[\theta_i|D] = \frac{1}{Z'} \prod_{j=1}^{m} \theta_{ij}^{(\alpha' q'_j - 1)}$$

which represents a Dirichlet distribution with a parameter α' and a parameter vector $\mathbf{q}' = (q'_1, ..., q'_m)$, that is

$$P[\theta_i|D] \sim Dir_{\alpha' \mathbf{q}'}(\theta_i)$$

The expectation of a specific variable at j dimension of θ_i is the mean posterior estimator of transition $s_i s_j$'s forward transition probability, denoted by $\hat{\theta}_{ij}$. We have

$$\hat{\theta}_{ij} = E(P[\theta_{ij}|D]) = E(Dir_{\alpha' \mathbf{q}'}(\theta_{ij}) = q'_j$$

$$= \frac{n_{ij} + \alpha q_j}{n_i + \alpha} \quad (8)$$

3.3 Setting the Prior

Our technique uses the information extracted from the Web logs to estimate α and $\mathbf{q}_i$. Let N be the sum of hits of all Web pages, and again let m be the number of states in the model. For each transition $s_i s_j$, we let $\alpha = N$. In setting the parameters q_{ij} of Equation 8, the prior outgoing transition probability of $s_i s_j$, we distinguish two cases. The first case is that s_j has incoming transitions. We set each q_{ij} to the normalized value of the average occurrence frequency of these transitions. (Normalization is necessary so the q_{ij} sum to one.) The second case is that s_j has no incoming transitions. In this case, the soonest we could observe a transition to s_j will be when the $(N+1)$st page hit occurs. We assume "optimistically" that the probability of this event is $\frac{1}{N+1}$, which would be the maximum likelihood estimate of s_j's frequency if the event did occur. Without knowing the order of previous page hits, we also assume that the transition to s_j is equally likely to come from any state. Therefore we set each q_{ij} to $\frac{1}{(N+1)m}$ divided by a normalization constant. For $I \geq 0$, let $s_1, ..., s_I$ be the states from which transitions to s_j were observed. Equation 9 gives the value of q_{ij} in each case:

$$q_{ij} = \frac{q'_{ij}}{\sum_{j=1}^{m} q'_{ij}} \quad (9)$$

where

$$q'_{ij} = \begin{cases} \frac{1}{I} \sum_{k=1}^{I} \frac{Occ(s_k s_j)}{Occ(s_k)}, & I > 0; \\ \frac{1}{(N+1) \cdot m}, & \text{otherwise.} \end{cases}$$

3.4 Updating the Model

Our model is updated in two cases. In the first case, estimated probabilities of existing transitions $s_i s_j$ in the model are updated, to account for new user-session data consisting of states s_i and s_j. (Both s_i and s_j represent existing Web pages.) Our technique applies recursive Bayes estimation [8] to update the model with new user-session data, using the new user-session data as the training data and the existing posterior distribution as the prior. The second case of updating the model is when states represented new Web pages are added to the model. Our model simply views a new Web page as an existing page that has never been visited. We used this option in our empirical study.

4 Empirical Study

To evaluate the effectiveness of our technique, we implemented it and performed an empirical study to evaluate the effectiveness of our technique. This section presents the study. We introduce the experiment subject (Section 4.1), describe the empirical setup (Section 4.2), and present and analyze the results (Section 4.3).

4.1 Subject of the Study

We used one month of log data for the NASA Web site as the subject for our study. The log data consists of 1,891,714 HTTP requests from real users. After filtering trivial sessions that consisted of only one user request, we constructed two types of user sessions: *non-error sessions*, which contained no requests with error status code, and *error sessions*, which contained requests with error status code.

Because the subject is a real-world application, we could not control the experimental environment and thus, did not know whether or not a user encountered a failure. Our only information was the requests that were not successfully handled by the Web server—the errors that were recorded in the Web log. Therefore, we used the error sessions retrieved from the Web log to evaluate the detection rate and used non-error sessions to evaluate the false-positive rate of our technique.

4.2 Empirical Setup

We implemented our technique in a tool called REBA (REcursive Byesian Analysis of Web Logs). REBA consists of a preprocessing component implemented in Jython 2.2.1 and a detection-model component implemented in Java 1.6.

In our implementation of REBA, we set the size of the sliding windows to 4 (see Section 2.3 for details), which covers the state to be checked and its three successive states (i.e., $w = 3$). Our observations indicate that after encountering a failure, many users make two or three more requests and then give up. Setting $w = 3$ means that to check whether a user encountered failures in a specific request s_i, our technique checks a sequence T of up to four requests including s_i. If a request is followed by fewer than three successive requests, the sequence T contains fewer than 4 requests. For example, if s_i has only one succeeding request (e.g., a user tried only one more request after encountering a failure) then $T = (s_i, s_{i+1})$.

We used the following process to perform the study using REBA:

1. Set the prior for estimating parameters. We used the first day's non-error sessions, consisting of 572 user sessions, to set the prior for estimating parameters in our model.
2. Train the model. We trained our model with the non-error sessions from the second day to the tenth day, which consisted of 2,404 user sessions.
3. Investigate the detection rate of the model. We tested our model using all error sessions of the remaining days of the month, which consisted of 500 user sessions. We checked the rate that these sessions are reported as error sessions.
4. Investigate the false-positive rate of the model. We tested our model using all non-error sessions of the remaining days of the month, which were 7,491 user sessions. We checked the rate that these sessions were reported as error sessions.
5. Update the model. In both Step 3 and Step 4, if a testing session was not reported as encountering errors, the model was updated using this session as new training data.

4.3 Results and Analysis

In this section, we present the results of the study and analyze the results.

Table 2: Results of detection rate and false-positive rate.

Detection Rate	False-positive Rate
0.60	0.11
0.70	0.26
0.80	0.40
0.90	0.53
1	0.87

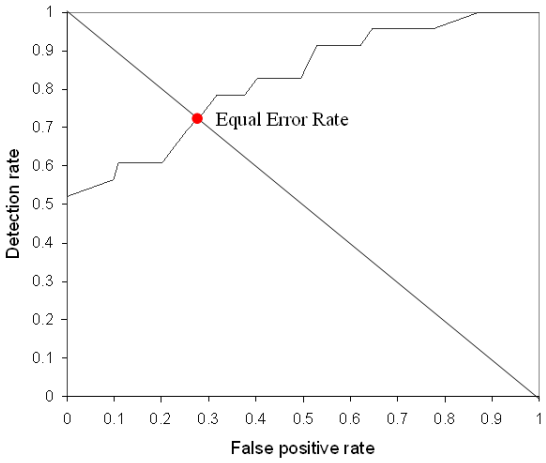

Figure 1: The ROC curve of model testing, wherein AUC is approximately 0.83 and EER is a detection rate of 0.71 with false-positive rate of 0.26.

4.3.1 Results

Table 2 shows several statistically important values of detection rates with false-positive rates. In this table, the first column is the threshold value, where $1 \leq w \leq 3$. The second column is the detection rate, and the third column is the false-positive rate. For example, the second column shows that the detection rate is 0.70 with a false-positive rate of 0.26.

4.3.2 ROC Curve Analysis

Figure 1 shows the *receiver operating characteristic* (ROC) curve of the study results. An ROC curve is a broadly-used graphical plot for evaluating and visualizing the performance of classifier models [9]. In the figure, the horizontal axis represents the false-positive rate, and the vertical axis represents the detection rate.

There are two important measures of an ROC curve. One measure of an ROC curve is the area under the curve (AUC). As a rough guide of how the AUC relate to the performance of a model, an area of 0.90–1 is excellent, 0.80–0.90 is good, 0.70–0.80 is fair, 0.60–0.70 is poor, and 0.50–0.60 is a failure. (See details at

http://gim.unmc.edu/dxtests/roc3.htm.) The other measure of an ROC curve is the Equal Error Rate (EER), which is used as the decision boundary when detection and false-positive have the same loss function. Geometrically, the EER is the intersection of the ROC curve and the diagonal from the top left to the bottom right of the graph. In Figure 1, the bold curve is the ROC curve and the AUC of this ROC curve is approximately 0.83; the intercept point of the ROC curve and diagonal is the EER, and the EER is a detection rate of 0.71 with a false-positive rate of 0.26. The analysis of ROC indicates that our model can detect failures with reasonable cost.

4.3.3 Result Discussion

We believe that the actual false-positive rate was lower than our empirical results. We evaluated the false-positive rate using non-error sessions whose status codes were not errors in Web logs. These sessions had no HTTP errors. However, user-visible failures caused by application failures, such as incorrect outputs or Web page display problems, are not recorded in Web logs. Thus, there could be other types of failures than HTTP errors in these non-error sessions. We expect that these non-error sessions contains failures and thus, the true false-positive rate is lower than our experiment results.

Our empirical results may also under-estimate the detection rate. The error session for evaluating detection rates were constructed based on the status code in Web logs. However, there are cases that an error status was recorded in Web logs but no failures actually occurred on the client side. For example, suppose an HTML template file is not used by a Web page, and the template file is removed from the Web site but the code of using the template still remains in the HTML code. If a user sends a request to this Web page, there is an error log entry of the template file in the Web logs, but no failure occurs on the client side. In this case, our study reported that a failure was encountered by users but was not detected by our technique. However, no failure actually occurred, and thus, the empirical results are lower than the real detection rate.

4.3.4 Robustness

In our study, our model was updated in both Step 3 and Step 4 if a testing session was not reported as encountering failures (Step 5 of the process we used indicates this updating). This process means that some error sessions were used to train the model because some error sessions were not reported as errors. Nevertheless, our model still detected failures with a reasonable false positive rate. This result demonstrates the robustness of the model.

5 Related Work

There are three approaches to detect failures in Web applications. The first set approach, which is the closely related work to ours and inspires ours, was presented by Bodik and colleagues [3]. Their technique addresses a special case of our technique, and our technique generalizes theirs and can detect more failures than theirs. Their technique detects failures by analyzing Web logs to find abrupt changes of hits on a Web page as an indication of users encountering failures. The technique assumes that users refresh their browsers when they encounter failures, and refreshing operations are recorded in Web logs as repeated requests to the same resource. However, this assumption does not always hold because users can have behaviors different from refreshing Web pages when they encounter failures. Our empirical results show that the frequency of users refreshing their browsers when they encounter failures is low: refreshing occurred in only 119 out of 500 error sessions. Like their technique, ours assumes that users' browsing behaviors change when they encounter failures. However, our technique does not presume any user's behavior. Rather, our technique constructs a first-order Markov model to compute the probability of a browsing path to infer whether a user's browsing behavior is anomalous.

The second approach applies statistical analysis to components' runtime information to infer anomalous execution paths (e.g., Chen and colleagues [6] and Kiciman and Fox [11]). Our technique differs from these in three ways. First, our technique can detect source code bugs that cause user-visible changes, which theirs is not always effectively detect failures caused by such software bugs [11]. Second, our technique is more efficient than these techniques. Ours does not require any instrumentation, but theirs rely on code instrumentation to monitor applications' runtime behavior. Third, our technique is more scalable. Their techniques can suffer scalability issues because the number of underlying components can grow exponentially as the systems grow. This growth occurs particularly when users' requests invoke a large number of objects of dynamic binding, which is an essential programming technique in multi-layer Web applications. In contrast, our technique requires only Web logs and is independent of the size of the application. However, theirs can detect failures that are not caused by software bugs such as performance failures, but ours does not ensure to detect such failures.

The third approach applies statistical analysis to infer failures that downgrade the performance of multi-tier Web applications (e.g., Barham and colleagues [2] and Cohen and colleagues [7]). Their statistical models analyze data from a variety of sources, including

networking, hardware, operating systems, databases, middle-ware, and application components. These techniques address failures that reduce the system's response. In contrast, our technique focuses on detecting application-level failures.

Another set of techniques are related to ours in that they also study user-browsing behavior. These techniques aim at such goals as optimizing Web caching (e.g. Padmanabhan and Mogul [14]), personalizing Web sites (e.g., Mobasher and colleagues [12], or studying user behavior patterns [1]. Our technique differs from these in two ways. First, these techniques aim at discovering the majority of user browsing behavior, whereas ours uncovers the minority—they discover patterns, but we find anomalies. Second, although both types of techniques use a fixed-size sliding window to cover $1 + w$ user requests, theirs uses the window in the $w + 1$ mode: using the w requests to predict the $(w + 1)$*th* request. In contrast, our technique uses a fixed-size sliding window in the $1 + w$ mode: using the w requests to estimate the one before the w requests.

6 Conclusion and Future Work

In our future work, we plan to improve our technique from three aspects. First, we need improve the detection power of our model. As shown in the ROC curve in Section 4.3, the slope of the false positive shows straight line segments, which indicates our model needs further sources of evidence for detection. A potential improvement of the model is to use a semi-Markov model to characterize the time that the user spends in each web page to infer additional indicators. We could also introduce more hidden states into the model to group users by their intention of using the Web site. Second, we need further evaluate our technique. In this work, we had only one cast study and could only use "hard errors" (i.e., errors reported by HTTP error code) as the "ground truth" to classify whether users encountered errors. However, the goal of the technique is to detect failures caused by software bugs that do not necessarily generate an error code. Users' responses to software bugs might be different from responses to hard errors, and such differences could potentially weaken the validate of the experimental results of the current work. However, it requires to have "known" data for improving the evaluation. Thus, we plan to collect user-interaction data of using multiple real Web sites in the controlled environment. One possible solution is to recruit user to use Web sites with seeded bugs and collect user-interaction data of using such Web sites. Third, we plan to introduce more human-computer interaction techniques into our model. For instance, our current model assumes an ideal environment where users perform their tasks without interruption. However, users may abandon their tasks by interruptions such as phone calls and emails. In such cases, our model could raise false alarms. A more sophisticated user model could help to handle more realistic users' environments.

References

[1] BALDI, P., FRASCONI, P., AND SMYTH, P. *Modeling the Internet and the Web: Prob. Meth. and Alg.* Wiley, 2003.

[2] BARHAM, P., ISAACS, R., MORTIER, R., AND NARAYANAN, D. Magpie: real-time modelling and performance-aware systems. In *9th Workshop on Hot Topics in Oper. Sys.* (May 2003).

[3] BODIK, P., FRIEDMAN, G., BIEWALD, L., LEVINE, H., C, G., PATEL, K., TOLLE, G., HUI, J., FOX, O., JORDAN, M. I., AND PATTERSON, D. Combining visualization and statistical analysis to improve operator confidence and efficiency for failure detection and localization. In *2nd Intl. Conf. on Automatic Computing* (June 2005), pp. 89–100.

[4] CARD, S. K., MORAN, T. P., AND NEWELL, A. *The Psychology of Human-Computer Interaction.* CRC Press, 1986.

[5] CHEN, M. Y., ACCARDI, A., KICIMAN, E., LLOYD, J., PATTERSON, D., FOX, A., AND BREWER, E. Path-based failure and evolution management. In *1st Symp. on Netw. Sys. Design and Impl.* (March 2004), pp. 309–322.

[6] CHEN, M. Y., KICIMAN, E., FRATKIN, E., FOX, A., AND BREWER, E. Pinpoint: Problem determination in large, dynamic Internet services. In *2002 Intl. Conf. on Dependable Sys. and Netw.* (June 2002), pp. 595–604.

[7] COHEN, I., GOLDSZMIDT, M., KELLY, T., SYMONS, J., AND CHASE, J. S. Correlating instrumentation data to system states: A building block for automated diagnosis and control. In *6th Conf. on Symp. on Operating Sys. Design & Impl.* (June 2004), pp. 231–244.

[8] DUDA, R. O., HART, P. E., AND STORK, D. G. *Pattern Classification*, 2 ed. Wiley-Interscience, 2000.

[9] FAWCETT, T. An introduction to roc analysis. *Pattern Recognition Letters 27*, 8 (June 2006), 861–874.

[10] HUBERMAN, B. A., PIROLLI, P. L. T., PITKOW, J. E., AND LUKOSE, R. M. Strong regularities in world wide web surfing. *Science 280*, 5360 (April 1998), 95–97.

[11] KICIMAN, E., AND FOX, A. Detecting application-level failures in component-based Internet services. *IEEE Transactions on Neural Networks 16*, 5 (September 2005), 1027–1041.

[12] MOBASHER, B., DAI, H., LUO, T., AND NAKAGAWA, M. Discovery and evaluation of aggregate usage profiles for Web personalization. *Data Mining and Knowledge Discovery 6* (January 2002), 61–82.

[13] OPPENHEIMER, D., GANAPATHI, A., AND PATTERSON, D. A. Why do Internet services fail, and what can be done about it? In *4th conf. on USENIX Symp. on Internet Tech. and Sys.* (March 2003), pp. 1–16.

[14] PADMANABHAN, V. N., AND MOGUL, J. C. Using predictive prefetching to improve world wide web latency. *Computer Communication Review 26* (July 1996), 22–36.

[15] RABINER, L. R. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE 77*, 2 (Feb 1989), 257–286.