Combining Visualization and Statistical Analysis to Improve Operator Confidence and Efficiency for Failure Detection and Localization

Peter Bodík[‡], Greg Friedman[†], Lukas Biewald[†], Helen Levine[§], George Candea[†], Kayur Patel[†]
Gilman Tolle[‡], Jon Hui[‡], Armando Fox[†], Michael I. Jordan[‡], David Patterson[‡]

[‡]UC Berkeley, [†]Stanford University, [§]Ebates.com

Abstract

Web applications suffer from software and configuration faults that lower their availability. Recovering from failure is dominated by the time interval between when these faults appear and when they are detected by site operators. We introduce a set of tools that augment the ability of operators to perceive the presence of failure: an automatic anomaly detector scours HTTP access logs to find changes in user behavior that are indicative of site failures, and a visualizer helps operators rapidly detect and diagnose problems. Visualization addresses a key question of autonomic computing of how to win operators' confidence so that new tools will be embraced. Evaluation performed using HTTP logs from Ebates.com demonstrates that these tools can enhance the detection of failure as well as shorten detection time. Our approach is application-generic and can be applied to any Web application without the need for instrumentation.

1. Introduction

Web applications are becoming increasingly complex and hard to manage. In particular, non-failstop application-level faults that cause user-visible failures are hard to detect without special case checks, yet they have a first-order impact on the user's experience that may result in temporary or permanent site abandonment. As much as 75% of time spent recovering from these failures is spent just detecting them [4]. Although a new focus on statistical anomaly detection and pattern recognition [10] promises to reduce the manual configuration and tuning required by current monitoring tools, statistical techniques invariably suffer from false positives (and sometimes false negatives), reducing the operator's confidence in the monitoring system.

Rather than ignoring this fundamental trust issue and removing the human from the loop, we believe a more promising path is an operator-aware division of labor for detecting such failures. To the computer we assign what the computer does best: statistical analysis of log data. For the human operator, we provide a tool to help bring her system experience and expertise to bear on interpreting and reacting to the alarms raised by the analysis engines; specifically, we provide rich visualizations of traffic information that allow her to quickly spot the sources of potential problems and to cross-check the reports of the statistical analysis tools. By taking advantage of the fact that human beings are excellent performers at visual pattern recognition, visualization helps the operator interpret failure alarms and identify their possible causes as well as keeping the effective cost of false alarms low by allowing her to rapidly identify them as such.

In determining what kind of analysis to perform on site logs, we observe that the site's end users are excellent "detectors" of site failures, in that their behavior typically changes when they encounter a malfunction. For example, if the link from the `/shopping_cart` page to the `/checkout` page is broken, users simply can't reach the `/checkout` page. Similarly, if a particular page does not load or render properly, users might click "Reload" several times to try to fix the problem. Since such behaviors are captured in HTTP logs, we can build statistical models of normal access patterns and then detect anomalies in user behavior. Since HTTP logs are application-generic, our approach can be applied to other Web applications without additional instrumentation.

Contributions

We present a visualization tool that allows operators to quickly spot anomalies or possible problems on their site in real time as well as confirm or investigate problem alarms reported by automated detection systems. To illustrate the latter ability, we apply some relatively well-known anomaly detection techniques to spot non-failstop failures in server logs from a real mid-sized Internet site Ebates.com; the basic insight is to look for anomalous patterns in end-users'

behavior as possible indicators of a failure. The information from these anomaly detectors is fed to the visualization tool, allowing the operator to visually inspect the anomaly in the context of previous and current traffic patterns and correlate the anomaly-score information to the traffic timeline. Unlike traditional visualization systems whose structure often reflects the architecture of the system, our tool visualizes metrics derived from users' site-visit behavior, which the site operators can more readily understand. We find that the use of the combined visualization and analysis tools would have allowed Ebates operators to detect and localize many actual site problems hours or days earlier than they actually did.

We make the following specific contributions:

- We use information-rich visualization to address the problem of operator trust in statistical learning algorithms. The synergy of visualization and automatic detection allows an operator to use *human pattern matching* to easily verify the warnings produced by our monitoring system.

- We monitor user behavior and automatically detect anomalies when the site is available but individual applications or functionalities are beginning to fail. This allowed us to quickly detect and help localize application-level failures from a real system: Ebates.com.

- Visualization of information was not based on systems architecture, which is the norm, but on metrics based on "black-box" user behavior. These user-oriented metrics offer greater insight into the status of the site and match our statistical algorithms. This match builds the trust relationship. Since our approach uses just HTTP logs to monitor user behavior, it can be used with any Web application.

Section 2 outlines our approach to combining visualization with automatic statistical anomaly detection. Section 3 describes the details of the algorithms themselves, our experimental setup and methodology, and evaluation metrics. Section 4 presents the results of applying our tools to real datasets, concentrating on the relative strengths and weaknesses of the different algorithms and the use of visualization to allow the operator to rapidly bring her experience and judgment into play to resolve ambiguities in failure reporting across the different algorithms. Section 5 discusses salient aspects of our results in light of our goal—helping the operator work more efficiently with automated detection techniques—and draws some tentative conclusions about implications for continuing work in this area. We then review some related work, outline possible future directions, and conclude.

2. Approach: Combining Anomaly Detection and Visualization

Our anomaly detection approach is relatively simple: we chose to look for abrupt changes in hit frequencies to the top 40 pages (which cover about 98% of traffic at Ebates). Fundamentally, this problem involves learning a baseline of hit frequencies, detecting deviations (anomalies) from that baseline, and determining the extent to which a "long lived" anomaly should influence the baseline (i.e., the sensitivity with which the baseline itself shifts in response to recent and/or anomalous data). Furthermore, when an anomalous frequency shift is detected, we need to *localize* the problem, i.e. determine which page(s) are most likely implicated as causing the anomaly.

To perform this analysis we use two statistical methods: Naive Bayes classification and the χ^2 (Chi-square) test. (The details of these algorithms are described in section 3). Other anomaly detection methods such as Support Vector Machines may perform better, but they don't allow us to easily determine which pages are the most anomalous. On the other hand, both Naive Bayes and χ^2 allow us to quantify the anomaly for each page.

While these techniques are useful for their ability to identify patterns in large amounts of data, as Hamming has said, "The purpose of computation is insight, not numbers." We augment the tools with *information visualization*, "the use of computer-supported, interactive, visual representations of abstract nonphysically-based data to amplify cognition", which has been shown to reduce the mental effort required to do search, recognition, and inference in connection with problem solving (see the first chapter of [2] for numerous examples).

The tool we built provides a compact and information-rich visual representation of: a) traffic to the 40 most requested pages, and b) transitions from and to these top 40 pages in user sessions. Figure 1 shows a screenshot of the main visualization interface of the tool. Instead of simply reporting `detected anomaly at 3:25pm`, the operator can immediately see the historic and current traffic patterns to confirm the anomaly. The tool also provides a simple interface for manual inspection of traffic patterns.

The emphasis on "live" interaction with the data distinguishes visualization from static graphical presentation. In our case, the operator can drill down on one of the visually-obvious traffic anomalies in Figure 1 to examine the page-transition rates during the anomalous interval, as shown in Figure 2.

The anomaly detection algorithms mentioned previously also feed information to the visualization tool. As we will describe, the algorithms report about once a minute on whether anomalous behavior was detected during that minute. To avoid bombarding the operator with alarms

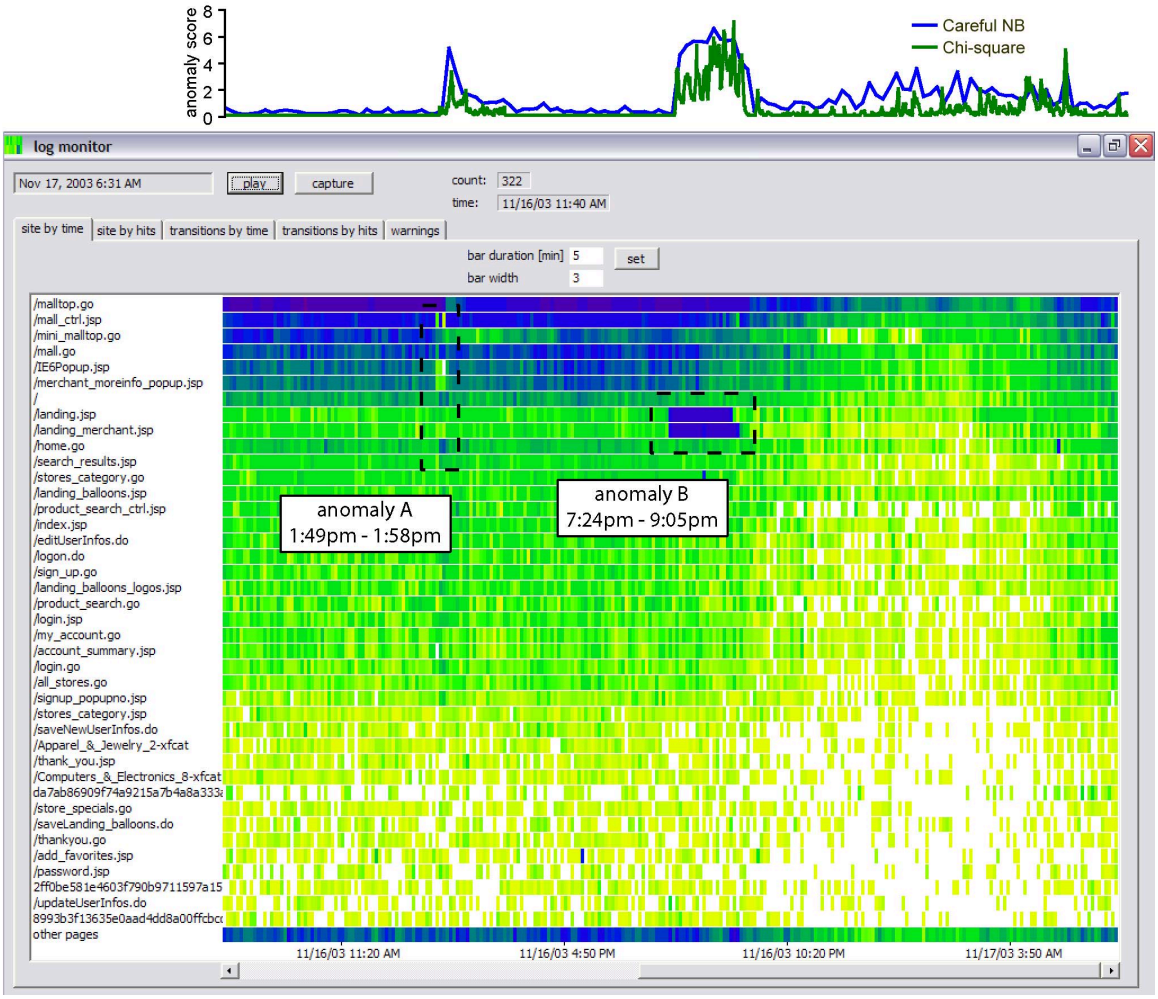

Figure 1. An annotated screen shot of the visualization tool. *(Note: if possible, figures 1, 2, and 3 should be viewed in color.)* The horizontal axis is time in 5-minute intervals. Each horizontal bar represents the hit count to one of the 40 most-requested pages; the bottom bar counts hits to all other pages combined. A blue tile means the corresponding page received $>$ 100 hits during that 5-minute interval; green $>$ 10 hits, yellow $>$ 1 hit, white (no tile) zero hits. The graph on the top shows the corresponding anomaly scores. In this screenshot we clearly see two anomalies from data set 1: A (1:49pm to 1:58pm) and B (7:24pm to 9:05pm).

for anomalies that persist over several minutes, consecutive anomalies after the first are grouped into a single warning. In addition, for each warning the tool reports the most anomalous pages, as scored by the anomaly detection algorithms, and the change in transition rates to and from the most anomalous pages; this information may help the operator localize the cause of the problem. For example, if the operator clicks on the "Warnings" tab in Figure 1 after selecting the anomaly marked as *B* in the figure, the Warning panel will display the following:

```
warning #2: detected on Sun Nov 16 19:27:00
start: Sun Nov 16 19:24:00 PST 2003
end:   Sun Nov 16 21:05:00 PST 2003

anomaly score = 7.03

Most anomalous pages:          anomaly score:
/landing.jsp                       19.55
/landing_merchant.jsp              19.50
/mall_ctrl.jsp                      3.69

change in transitions FROM /landing.jsp
page:                         before:   after:
/landing_merchant.jsp         37.13%    93.17%
/mall_ctrl.jsp                21.85%     0.83%
```

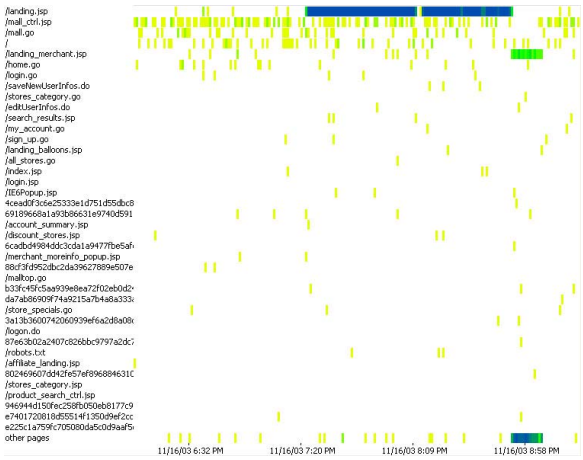

Figure 2. Page transitions from `/landing_merchant.jsp` page during the anomaly B in data set 1, in 1-minute intervals. A sudden increase of transitions to `/landing.jsp` (the topmost row) at 7:24pm is evident from the figure. This represents an alternate view of part of the time period from Figure 1.

3. Test Data and Methodology

In this section we describe the HTTP logs we analyzed, give details of the analysis algorithms, and describe our evaluation methodology before proceeding to experimental results.

3.1. HTTP Access Logs

A typical three-tier Internet application consists of a tier of Web servers, a tier of application logic servers, and a tier of persistent storage servers. Popular Web servers include Apache and Microsoft IIS; application servers may be framework-based, such as Java 2 Enterprise Edition (J2EE) servers, or custom-written in-house; persistence may be provided by a database such as Oracle or MySQL or by a file server.

Ebates.com provided us with 5 sets of (anonymized) access logs recorded by the Web server tier. Each set consisted of HTTP traffic to 3 Web servers during a contiguous time period ranging from 7 to 16 days. Each period contains at least one web application failure as well as significant intervals of "normal" application behavior. The access logs contained the following information about every user request: Apache server's time stamp, local URL of page accessed, URL parameters (parts of the URL passed as parameters to scripts on active pages), session ID, application server that served the request, and anonymized user ID.

3.2. Analysis Using χ^2-test

Intuitively, we might expect that under normal conditions, vectors of page hit counts collected during different time intervals should come from the same distribution, or more generally, a vector of hit counts collected during the current time interval should come from the same distribution as the "historical norm." The χ^2-test [12] can be used to compute the probability that two data vectors $A = (a_1, \ldots, a_n)$ and $B = (b_1, \ldots, b_n)$ come from different distributions; it has been used, e.g., to detect anomalies in network traffic [14].

The test is performed in the following way:

1. Let $S_a = \sum_{i=1}^{n} a_i$, $S_b = \sum_{i=1}^{n} b_i$, and $s_i = a_i + b_i$.
2. Compute the *expected value* for each a_i and b_i: $E_i^A = s_i S_a/(S_a + S_b)$, $E_i^B = s_i S_b/(S_a + S_b)$.
3. Compute the total χ^2 value of the two vectors: $\chi^2 = \sum_{i=1}^{n}(a_i - E_i^A)^2/E_i^A + (b_i - E_i^B)^2/E_i^B$.
4. Compute the significance s of the test using the χ^2 distribution with $n - 1$ degrees of freedom.
5. Finally, an anomaly score is computed as $-log(1-s)$.

From the access logs, we first identify the N most popular (by hit count) pages of the site (we used $N = 40$). We compute the vector $C = (c_1, \ldots, c_{40})$ of hit rates to each of these pages during the current time interval and compare it to the "historically normal" vector $H = (h_1, \ldots, h_{40})$ of hit rates for the same pages. Since different types of traffic anomalies may take different amounts of time to become evident, we consider time intervals varying in length from 1 to 20 minutes. Since by definition the χ^2 test is not valid if each page didn't receive at least 5 hits, we exclude pages with fewer than 5 hits. The actual algorithm, run once per minute, is as follows (let t be the current time):

1. compute the historic traffic pattern H over all previous data, *excluding* time intervals marked as anomalous. (Initially, we assume all intervals are normal.)
2. for every $t_0 \in \{1, 2, \ldots, 20\}$:
   (a) compute current traffic pattern C from time interval $\langle t - t_0, t \rangle$
   (b) compare C and H using the χ^2-test. If the significance of the test is higher than 0.99, mark the interval $\langle t - t_0, t \rangle$ as anomalous.

When a period is declared anomalous, we assign an anomaly score to each page based on that page's contribution to the total χ^2 value: $((c_i - E_i^C)^2/E_i^C + (h_i - E_i^H)^2/E_i^H)$.

We also detect significant changes in page transitions that occurred when the anomaly started. We compare the traffic *before the anomaly* (time interval $\langle t_0 - t, t_0 \rangle$, where t_0 is the start of the anomaly) and *during the anomaly* ($\langle t_0, t_1 \rangle$, where t_1 is current time). Thus, every minute of an anomalous traffic we compare the transitions to and from the top 40 pages before and during the anomaly using the χ^2-test.

3.3. Analysis Using Naive Bayes Classifier

The second type of analysis involves training a Naive Bayes classifier [6] to detect anomalies. Again, we use the access logs to compute the hits per unit time to each of the top N pages on the site, $(c_1, \ldots, c_N)$, during the current time interval. The c_i's are normalized by dividing by the total hit count during the interval so that they are in the range of 0 to 1. We also compute the combined hit frequency to all remaining pages on the site and the difference in total hit count between the previous time period and the current one. By using the Naive Bayes model we make an (incorrect) simplifying assumption that all the 42 (=$N+2$) features are conditionally independent. However, Naive Bayes is very often successfully used in practice even though this theoretical requirement is rarely satisfied.

We divide time into 10-minute time intervals and use this classifier to determine whether the current time interval is *normal* ($S = s^+$) or *anomalous* ($S = s^-$). The conditional probability of each feature $\mathbf{f}_i$ given $S = s^+$ is modeled by a Gaussian distribution whose mean μ_i and variance σ_i^2 are estimated for each feature using maximum-likelihood estimation from the previous time intervals.

If we knew *a priori* which time intervals were anomalous and which were normal (in the parlance of machine learning, if we had labeled data), it would be trivial to calculate the mean and variance of $p(\mathbf{f}_j | S = s^+)$ and $p(\mathbf{f}_j | S = s^-)$ using maximum likelihood estimation (MLE). However, as is typical for real systems, our data is unlabeled (i.e. we don't know which periods are anomalous) so we have to do unsupervised learning. In the absence of labeled examples of s^-, we choose to model the conditional probability $p(\mathbf{f}_j | S = s^-)$ using a uniform distribution over the range of possible values (i.e., 0 to 1).

The standard approach of using Expectation Maximization (EM) to simultaneously learn the value of s and $p(\mathbf{f}|s)$ is too slow for real-time use on high-volume data, so we approximate it with two separate methods:

- Unweighted learning (*Eager NB*): We estimate μ_i and σ_i^2 with the assumption that every previous time interval is *normal*, i.e. we "label" every previous interval as $S = s^+$. This is a reasonable first-order assumption as long as the majority of states are in fact normal, in other words, as long as failures are rare. However, if an anomaly occurs *and persists*, this technique will quickly "adjust" to the anomaly by treating it as normal. We therefore call this an "eager" learner.

- Probabilistically-weighted learning (*Careful NB*): In estimating μ_i and σ_i^2, we weight each past time interval by the probability that it is *normal*. Thus, the more anomalous a time interval appears, the less it is incorporated into the model of normal behavior. This method will continue to detect an anomaly even when it persists for a long time, but if the long-lived "anomaly" is in fact a new steady state, it will take longer for this method to adjust to it. We therefore call this a "careful" learner.

In our Naive Bayes approach, we don't learn the prior probabilities of *normal* and *anomalous* time intervals. Instead, we use the priors as a parameter that trades off between a low false positive rate (for low $Prob(anomalous)$) and a high anomaly detection rate (for high $Prob(anomalous)$).

The classifier reports an anomaly score for each time period; this score is calculated as $-log(Prob(\mathbf{f}|normal))/n$, where n is the number of features used. To localize the most likely features that caused the anomaly, we assign an anomaly score to each feature $\mathbf{f}_i$ as $-log(Prob(\mathbf{f}_i|normal))$.

Note that in many cases, the judgment of whether a long-lived anomaly is in fact a new steady state may require operator intervention. Therefore, rather than trying to automatically determine this, we allow the operator to visualize both the raw data and the anomaly scores over time reported by each algorithm.

3.4. Methodology

The logs we received were collected in the past, and the failure incidents reflected in them had already been diagnosed and dealt with. Our methodology therefore consisted of the following steps:

1. With little or no knowledge of what events occurred in a data set, run our detection algorithms. For each data set, the models were initialized as untrained at the beginning of the data set and learned in an online fashion.

2. For each anomalous period (and some normal regions as well), examine the traffic patterns during that period using our visualization tool in conjunction with the graphs of anomaly scores reported by our anomaly detectors.

3. Use the visualizations and graphs to discuss each reported incident with the CTO and operations engineers at Ebates to reconstruct "what really happened" during that incident as completely as possible.

4. Based on these discussions, classify each reported anomaly as either a true positive, a false positive (clearly attributable to a non-fault event, such as a failure-free update to the Web site), or a possible false positive (one we could not attribute to an event, or more often, that we could attribute but we could not unambiguously determine whether or not the associated event was a failure).

5. Based on these discussions, determine how much sooner a site failure could have been detected or prevented had our tools been in place at the time of the incident.

3.5. Evaluation Metrics

Traditionally, failure detection is evaluated in terms of precision and time-to-detection. Precision is defined as true positives divided by all positives, i.e., $TP/(TP + FP)$, where true positives are the number of actual failures detected and false positives the number of identified events that are *not* failures. However, in dealing with real data from a complex service, these metrics are problematic.

First, part of the motivation behind our work is precisely that existing detection techniques *do not* detect certain kinds of fail-stop failures. Hence, in cross-checking our results against the best knowledge available to the operations staff, we do not have 100% certainty that the list of known failures is exhaustive. Second, some false positives are attributed to non-fault events that result in a change in user behavior, or events that cause bona fide performance anomalies that are nonfatal under moderate load (for example) but lead to user-visible failure under heavier load. To be conservative, we count such incidents as "false positives" in our evaluation. Finally, the notion of "time to detect" presupposes that there exists an instant in time before which there was no failure and after which there was unambiguously a failure. For non-failstop failures, especially those that manifest only under increased load, it is unclear how to choose this moment. Furthermore, even if we could assume that the time of fault occurrence was well-defined, we lack the ground-truth knowledge to establish conclusively what that time was.

The information we do have includes the time a particular failure was in fact detected by existing techniques (whether automatically or manually by Ebates staff) and the ostensible cause of the failure (localization information) as determined by Ebates staff. When we measure true and false positives, then, we assume the best case corresponds to detecting all failures detected by Ebates staff and misclassifying no other events. We also measure *advance warning*—the amount of extra reaction time the staff would have had because our tool identified a potential failure before the staff did, and whether this advance warning might have helped mitigate or avoid a failure. Finally, we ask the question: had the staff been given the localization information provided by our algorithms, how useful would it have been in identifying the cause of the failure? In our measurements we report the qualitative answer, on a scale of 1 (not useful) to 10 (vitally useful) given by Ebates staff. (We did not collect localization score data on one of our algorithms, *Eager NB*.)

4. Discussion of Results for Each Data Set

In this section we discuss the results of combining anomaly detection and visualization on each of the datasets we examined. We then summarize the overall performance of our techniques, paying particular attention to the differences in each algorithm's ability to detect particular failures, the role of visualization in helping the operator sort out "what really happened", and the implication for combining both types of operator information.

4.1. Data Set 1: Account Page

The major problem that occurred during this week was a site slowdown (and then crash) caused by the *account page*. The bad account page had existed for a long time, but this time period saw a larger-than-ever hit rate to the page as customers logged in to check their accounts following a quarterly mailing of rebate checks to customers. The problem was first detected by Ebates on Monday (Day 3 of the data set) at approximately 6:23am, diagnosed by Ebates at approximately 12 noon, and the offending page removed at about 12:30pm. The problem reappeared on Day 4 from 5:38am to 7:13am as Ebates staff tried putting the page back up, but took it down again. On Day 6 starting about 8pm, there was massive quality assurance of this page against 2 servers on the live site to confirm the fix. The fix was verified on Day 6 at 11:21pm. Given this chronology reconstructed from Ebates' information, we now compare it with the behavior of our algorithms.

Day 2. We detected a mid-size anomaly (*A*) on Day 2 at 1:49pm and a significant anomaly (*B*) on the same day from 7:24pm until 9:05pm, centered around two pages not directly related to account activity. The number of hits to these pages increased from less than 5 every minute to about 50 hits a minute. Anomaly B can be seen on Figure 1. After zooming in on this anomaly and switching to the *transitions view* (Figure 2), we can see a significant change in the pattern of pages an HTTP session visits immediately after visiting `/landing_merchant.jsp`. This can be potentially very important information for the operator. We later learned that these two anomalies corresponded to database alarms raised by Ebates' database monitoring system as a result of a quarterly batch job that placed a

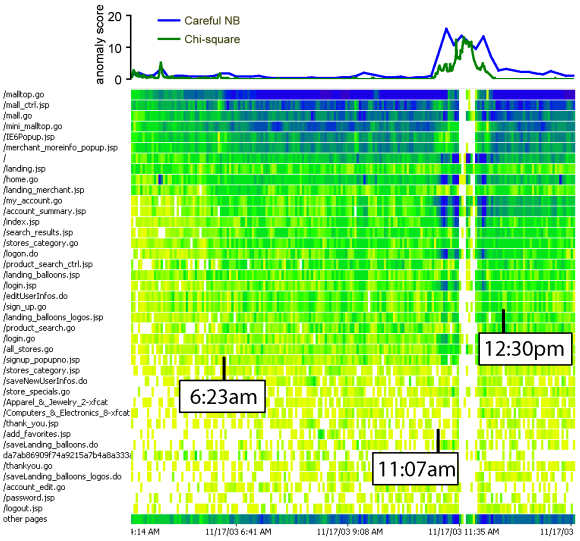

Figure 3. Visualization of the anomaly followed by a site crash on Day 3 of data set 1. An (undiagnosed) problem was first detected by Ebates at 6:23am; Ebates diagnosed the problem and removed the buggy account pages at about 12:30pm. Our algorithms detected and localized the first anomaly at 11:07am.

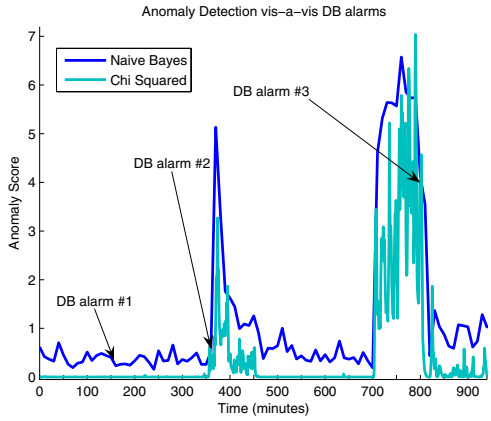

Figure 4. Anomaly scores and database alarms on Day 2 of data set 1. Note that we did not detect the first database alarm as an anomaly.

heavy load on the database. Figure 4 shows both our algorithm's anomaly scores over time, and the occurrence of three database alarms at Ebates, for a 16-hour period in this data set. In particular note that we detected an anomaly at 7:24pm (approximately time 700 in the graph), about 100 minutes before the third database alarm was raised. Although the pages reported most anomalous are not directly related to the account page, Ebates staff said that knowing about this anomaly would have put the site operators on alert for possible problems, and may have led them to detect the problem earlier than they did.

Day 3. The next anomaly, the largest in the data set, starts at 11:07am on Day 3. This anomaly is presented in Figure 3. It can be seen (and it was correctly reported by the NB algorithm) that two of the most anomalous pages are the account pages. At this point, Ebates staff was aware of a performance problem, but they would not discover its cause until 50 minutes after our anomaly warning, which reported the two account pages as the most highly anomalous. Ebates indicated that this would have been a strong diagnostic hint to the operators and would likely have reduced diagnosis time.

The last significant anomaly we detected, 7:51pm on Day 6, was due to an intensive stress-test effort performed on the live site to validate the fix.

All three algorithms were also affected by significant night anomalies lasting from approximately midnight to 2am on most nights. In fact, we saw these night anomalies to varying degrees in all five of our data sets. For example, you can notice the increased anomaly score around this time period on Figure 1. We later learned that these may have corresponded to a cache warm-up effect resulting from a nightly cache flush (of which some staff were unaware). There are other possible explanations for the night anomalies, including simply higher variance due to a lower number of total hits. We intend to further investigate this phenomenon in the future.

In summary, the anomaly and diagnostic information available in this case would have been helpful in diagnosing the cause of a poorly-understood performance problem and would have called attention to the account page problem an hour before it was identified.

4.2. Data Set 2: Landing Loop

This data set included a site crash due to a bug whereby a "landing page" (an entry page to the site) was incorrectly redirecting users back to the landing page itself, resulting in an infinite loop that eventually brought down the site. All 3 algorithms (*Careful NB*, *Eager NB*, and χ^2) detected significant anomalies 2 days prior to Ebates' detection of a major problem and $2\frac{1}{2}$ days before the crash, corresponding to the introduction of two new "landing pages" to the site. (Note that the bug may have been introduced at this time or at a point in the future.) All 3 algorithms also detected significant (and increasing) anomalies in the site beginning several hours before the site crashed. *Careful NB* and *Eager NB* provided localization information that the CTO rated as an 8, on a scale of 1-10, for the usefulness this would have had in detecting and diagnosing the problem so as to avoid

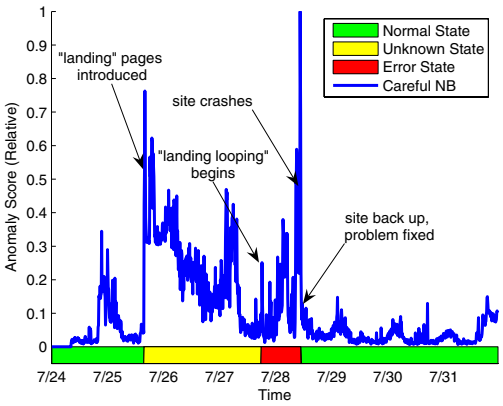

Figure 5. Anomaly scores over time for *Careful NB* on data set 2, "Landing Loop". The time period from the introduction of the landing pages until Ebates' detection of the landing loop problem is marked as an unknown system state, since we haven't been able to determine with 100% certainty whether this time period was problematic for Ebates or not.

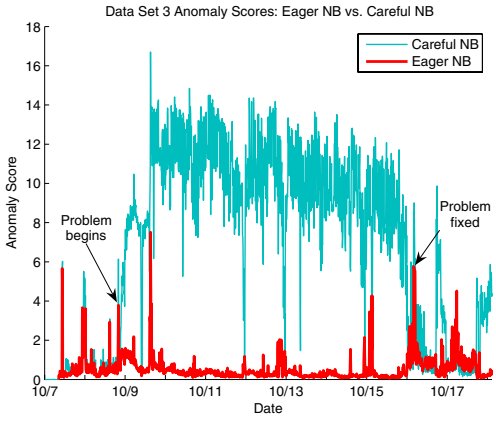

Figure 6. Whereas "careful learner" *Careful NB* continues to detect the broken signup page anomaly (data set 3) for all 7 days, "eager learner" *Eager NB* quickly decides that this new behavior is normal.

the crash. χ^2 had one (possible) false positive: the early detection $2\frac{1}{2}$ days before the crash. *Eager NB* and *Careful NB* each had two (possible) false positives. According to the CTO, even if the initial detection $2\frac{1}{2}$ days before the crash did not represent the onset of the major bug, the warning and localization information provided at that time would have been very helpful in diagnosing the problem when the major failure did start to occur.

Figure 5, which shows the anomaly scores over time for *Careful NB* for this data set, illustrates the effect of careful learning. At the introduction of the problematic pages, *Careful NB* detects a significant change in the distribution of the traffic to the site. Because this raises the anomaly score significantly, *Careful NB* weights this time period extremely low in incorporating it into its model of "normal". Traffic characteristics to which *Careful NB* is sensitive remain anomalous for the next two days, so *Careful NB* is very slow to decide that this is in fact "normal" behavior. In contrast, as shown in Figure 6 (which represents data set 3), the "eager learner" *Eager NB* quickly concludes that a new and different traffic pattern is no longer anomalous, due its assumption that the behavior of all previous time periods should contribute equally to the profile of "normal" behavior. The third algorithm, χ^2 (shown in Figure 7 for data set 5), works by detecting changes in traffic patterns over relatively short time periods, and therefore exhibits much more bimodal behavior than *Careful NB*.

4.3. Data Set 3: Broken Signup

This data set does not contain a crash, but rather an incomplete deployment of a "new user sign up" page which had gone undetected by Ebates operators. Served to new users for over a week, this page displayed a blank page with an error on it instead of the expected site content, rendering it impossible for new users to come to the site. The problem did not affect existing users.

Careful NB and χ^2 detected the introduction of the problem 7 days before Ebates' diagnosis. (*Eager NB* detected an anomaly at this time too, but it was close to its noise threshold, so we do not count this as a detection for *Eager NB*.) *Careful NB* provided localization information at problem introduction time, and during the entire 7 day anomaly, that Ebates said would have been extremely helpful in localizing the problem. Because of the length of the anomaly, *Eager NB* and χ^2 began to view the anomalous period as normal, but *Careful NB* (the "careful learner") continued to report an anomaly for the entire 7 days, as seen in Figure 6. This is another example of a case in which the operator's understanding of the system would help resolve the ambiguous result of applying algorithms that are sensitive to different timescales.

4.4. Data Set 4: Badly Optimized Page Bug

In data set 4, a new page that invoked an inefficient database query caused a site crash when its hit rate increased as a result of a large email campaign. The resulting database overload subsequently led to site failure. *Careful NB* and *Eager NB* detected the anomaly 4.5 hours before

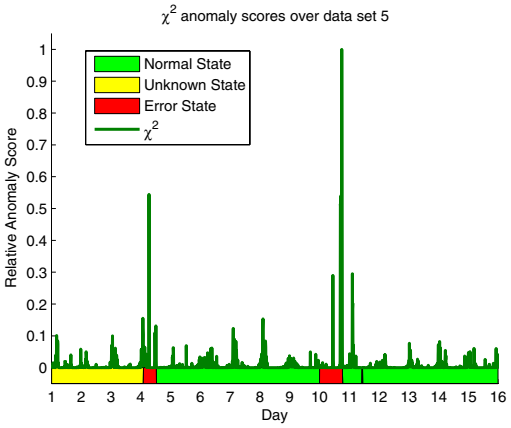

Figure 7. Anomaly scores over time for χ^2 on data set 5, "Bad URL and Runaway Query". The major failures occured on days 4 and 10.

the crash (3 hours before Ebates staff detected the problem). The anomaly level was initially low, increasing as the problem escalated. χ^2 detected the problem 3 minutes after Ebates staff detected the problem. Ebates staff believe they could have avoided the crash if they had had the anomaly localization information provided by *Careful NB*.

4.5. Data Set 5: Bad URL and Runaway Query

Data set 5 contained 2 significant failures. The first failure due to a configuration error in which the 'shopping url' for one of Ebates' merchants was inadvertently mapped to a popular search engine site. All Ebates users who followed that link were instead sent back to Ebates where shopping sessions were generated continuously and in rapid succession until the site crashed.

χ^2 detected this failure 5.5 hours before Ebates could diagnose the problem. Both NB algorithms detected the problem at the same time, but the anomaly scores were within the noise level, so we don't count this in our results as a detection for the NB algorithms. However, once again Ebates staff said the localization information provided by the NB algorithms would have been a tremendous diagnostic aid.

The second significant failure was a runaway query causing significant database performance problems. All three algorithms detected this failure concurrently with Ebates' detection. Figure 7 illustrates the χ^2 algorithm's behavior over this data set.

4.6. Summary of Results

Table 1 summarizes the overall results of our analysis. For five of the six major faults in our log data, at least one of the algorithms detected the problem as well as provided useful localization information prior to Ebates being able to diagnose the problem; the sixth was detected concurrently with its occurrence. Our algorithms performed less well on the seven minor faults (four database alarms, one brief outage associated with a code push, one brief re-introduction of a buggy page, and one massive QA effort on the live site to verify a fix). Of the three missed faults (false negatives), two were database alarms that may not have had any discernible impact on users and the third was a brief introduction and then removal of a buggy page. The three known false positives were all failure-free code updates to the application. Note that we did not consider the predictable nightly anomalies as false positives, since they were easily filtered out by time. We did not perform localization or advance-warning analysis on the minor faults.

Table 1. Summary of our results over all 5 data sets. For major faults, χ^2 had a higher detection rate and fewer false positives than *Careful NB* or *Eager NB*, but *Careful NB* provided more useful diagnostic information.

Major fault	*Careful NB*	*Eager NB*	χ^2
Faults Detected	5/6	4/6	6/6
Known FP's	1	1	1
Possible FP's	3	3	2
Detection rate	83%	67%	100%
Precision	56-83%	50-80%	67-86%
Local. score	8.6/10	n/a	4/10
Minor faults	*Careful NB*	*Eager NB*	χ^2
Faults detected	4/7	4/7	4/7
Known FP's	3	3	0
Possible FP's	5	4	2
Detection rate	57%	57%	57%
Precision	33-57%	36-57%	67-100%

Table 2 summarizes our results broken down by data set. For each data set we show number of major and minor faults detected (out of the total number of total major and minor faults respectively); number of known false positives and possible false positives; the advance warning time (AWT) for major faults, i.e. how long from the time our algorithms detected an anomaly until Ebates' initial localization of the related fault; and the localization score, the usefulness of diagnostic information as estimated by Ebates staff on a scale of 1 to 10, lowest to highest, for the major faults. In data set 2 we are not certain whether if that detection represents the actual onset of the problem (see section 4.2); if not, the detection in this data set was approximately concurrent with Ebates' detection.

Table 2. Performance by data set, using χ^2 for detection and *Careful NB* for localization.

Measure	DS1	DS2	DS3	DS4	DS5	Total
Major faults	1/1	1/1	1/1	1/1	2/2	6/6
Minor faults	3/5	0/0	1/1	0/0	0/1	4/7
Known FP's	0	0	1	0	0	1
Possible FP's	0	1	1	1	1	4
AWT	1h	50h?	7d	0m	5.5h,0m	avg: 37h
Local. score	8	8	9	10	8	avg: 8.6

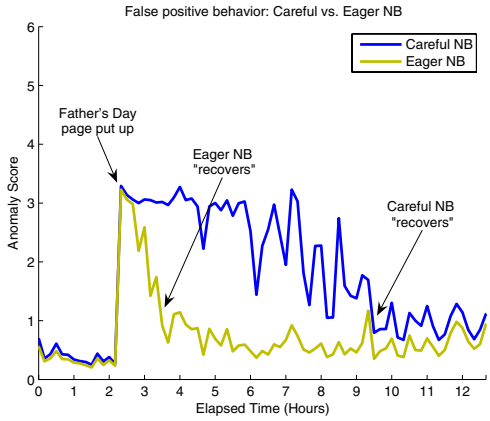

Figure 8. Differing behavior of Careful vs. Eager learning in handling false positives. Careful NB takes 9 hours to "recover" from the false positive, while Eager NB reocovers in about 90 minutes.

5. Discussion: Role of the Operator

5.1. Classifying False Positives

In our experience, support from operations staff is essential for asserting the ground truth of a warning system. In our case, we needed their help to interpret the failure data and verify the conclusions of our anomaly detectors; but in general, dealing with false positives may require operator intervention, since the operator(s) may be best qualified to determine whether an anomaly, transient or long-lived, is a true problem or a false alarm. Figure 8 illustrates the tradeoff between early detection and false positives: the introduction of a new and bug-free "Father's Day" page to the site caused both *Careful NB* and *Eager NB* to declare an anomaly; *Careful NB* took 9 hours to decide that this new page was not problematic, while *Eager NB* reached this conclusion in only 90 minutes. Similarly, an anomaly we detected several hours before the anomalous pages caused a site crash might be dismissed by the operator since at the time Ebates had just sent emails to all its customers, stimulating increased site activity; but if anomalous behavior persisted, the operator might realize that the localization information could help drill down on the problem and determine if there was really cause for concern.

5.2. Detecting Different Types of Anomalies

Naive Bayes and χ^2 respond to different kinds of changes in traffic patterns and are useful for complementary tasks. Naive Bayes is sensitive to *increases* in frequency to *infrequent* pages. Consider a page that normally accounts for only 0.1% of hits to the site with variance 0.01%: since Naive Bayes models each page's hit frequency as a Gaussian distribution, an increase in that page to, say, 5% of page hits is modeled as extremely improbable. In our datasets many failures were related to bugs in relatively infrequently-accessed pages, which made NB an effective diagnostic tool.

In contrast, the χ^2 test is robust to changes in hit counts to unpopular pages, in part because the validity of the test requires excluding any page not receiving at least five hits in a given time interval. This results in χ^2 being more sensitive to increases and decreases in *frequent* pages, but less useful for precise localization: its bias toward frequently-accessed pages often caused it to report those pages as the most anomalous, but that is not useful since frequently-accessed pages are generally going to be affected at some point by most failures anyway. For this reason, the summary of per-dataset results reported in Table 2 rely on χ^2 for detection and Naive Bayes with probabilistically-weighted learning (*Careful NB*) for localization.

The difference in behavior between the techniques can be seen in Figure 9, which represents data set 4. NB is able to detect this anomaly 3 hours before χ^2, because it detects an increase in hits to very infrequent pages. These pages continue to increase in frequency, and 3 hours later they end up having a major deleterious effect on the site. This causes changes in hits to highly frequent pages, which χ^2 then detects. Thus, the determination of which algorithm detected the anomaly tells the operator something about the *nature* of the anomaly, and with experience, an operator could learn to recognize "patterns" such as this one, in which detection by one algorithm is later followed by detection by another. Again, our contribution is the pervasive use of visualization to take advantage of the operator's experience with and understanding of the system as well as her ability to rapidly absorb visually-presented information.

5.3. Reconstructing the Ground Truth

To precisely compute the accuracy, time to detect, and false positive/false negative rates of our techniques, we need to know the "ground truth" of exactly what problems oc-

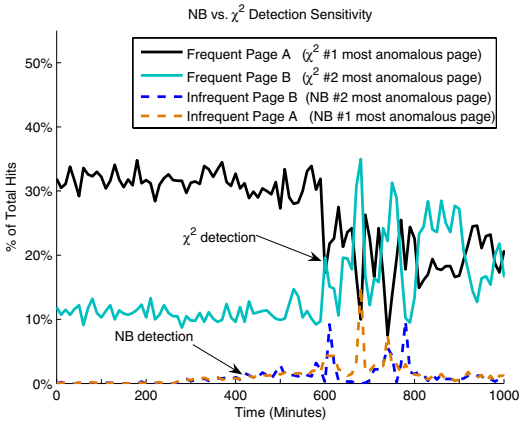

Figure 9. Illustration of differences in detection sensitivity of NB PWL and χ^2. NB PWL is more sensitive to increases in hits to infrequent pages, which makes it able to detect changes earlier and seems to make it a better localizer. χ^2 is more sensitive to changes to hits to more frequent pages, making it less prone to false positives.

curred, when they were introduced, and when they were detected. Reconstructing this information required the cooperation of the operations staff, who even after going through system monitor logs, email archives, and chat logs could only partially reconstruct some of the incidents. We still don't know the precise time some failures were introduced (only when they were detected by Ebates staff), and we still cannot determine whether three of the anomalies we detected really corresponded to a transient (but undetected) site problem or were false positives. This in turn can make it difficult to determine whether an anomaly detected prior to an actual failure was an early warning signal or an unrelated false alarm. We believe that this is an inherent problem of working with real-world failure data. Thus, instead of reporting "time to detect" and "false positive rate", we present alternative evaluation metrics based on the amount of advance warning provided by our tools relative to existing detection techniques used at Ebates.

5.4. Generalizing our Approach

In this work we detect problems with Web applications by detecting anomalies in the HTTP traffic to the site. These anomalies can be caused by users when accessing a failing service or by the service itself when requests from users are forwarded incorrectly. We obtained the results described in Section 4.6 by analyzing all HTTP traffic data from 3 web servers in Ebates.

In order to build models of normal behavior and detect deviations from that model, one needs a representative sample of the HTTP traffic to a site. Additionally, there needs to be enough such traffic so that statistical techniques can be effectively applied. To investigate the behavior of our algorithms with smaller sample sizes, we performed experiments where we randomly subsampled the HTTP log and analyzed the first three days of data set 1 (containing three anomalies as described on page 6). By using just 5% of all available data (approximately 12 thousand requests a day), we still obtained the same results as when using all traffic from the 3 web servers.

In our Ebates experiments, we model the frequencies of the 40 most frequently accessed pages on the web site. Even though the Ebates users hit several thousand distinct pages each week, hits to just the top 40 pages covered about 98% of all traffic. We decided to model just these 40 pages under the assumption that any significant problem would affect these pages relatively quickly. For web sites with a broader distribution of hits, there should be no difficultly in modeling more pages as necessary.

In summary, we make very few assumptions about the behavior of users and believe that this approach can be applied to an arbitrary Web application.

6. Related Work

Web Usage Mining (as described in [13], for example) is a large and active field. These techniques have been applied to a variety of areas, but we believe we are the first to explore their use in application failure detection and diagnosis. Kallepalli and Tian [9] employ a form of hidden Markov Models to learn user behavior from web logs, but their focus is on building an effective test suite rather than real-time problem detection and localization. Commercial products for monitoring applications and their software and hardware platforms, such as IBM Tivoli [8], HP OpenView [7], and Altaworks Panorama [1], focus on system and internal application metrics rather than changes in user behavior to discover failures.

Pinpoint [3, 11] employs anomaly detection methods for problem detection and root-cause analysis in distributed applications. While their approach involves instrumenting the application and/or middleware to discover patterns in application component dependencies, our approach uses only generic HTTP logs and treats the structure of the application as a black box.

Cohen et al. [5] have applied an augmented type of Bayesian network to the problem of metric attribution: determining which low-level system properties (CPU load, swap activity, etc.) are most correlated with high-level behaviors such as performance problems. Like us, they selected a Bayesian network because its property of interpretability allows the network to be used for localization.

7. Future Work

Ebates has expressed interest in deploying our visualization tools on live data and in working with us to develop a better "ground truth" for evaluation. However, our experience suggests that a 100% accurate ground truth might be unrealistic, so we are in the process of obtaining similar data sets from two other companies to repeat our experiments.

We want to incorporate real-time feedback from the operator into our models and produce more informed warnings. For example, if the current long-term anomaly represents a normal behavior, the operator should be able to specify that this is actually *the new normal behavior*. On the other hand, if the recent anomaly represented a normal behavior (e.g., page update), we shouldn't report a warning the next time a *similar* anomaly appears. Also, we are currently extending our models of web traffic to capture the correlations between the frequencies of pages.

8. Conclusion

Notwithstanding the promise of statistical analysis techniques for detecting and localizing Internet service failures, the judgment of experienced operators can help disambiguate conflicting warnings, resolve apparently spurious warnings, and interpret problems flagged by such algorithms. We showed that visualization combined with anomaly detection and localization can help human operators bring their expertise and experience more efficiently to bear on such problems, reducing detection time, diagnostic effort, and the cost of classifying false positives.

In particular we detected anomalies in user traffic to a real mid-sized Internet site using Naive Bayes and the χ^2-test. Our techniques detected four out of six failures more quickly than the site's staff, and the visualization helped to understand the types and sources of anomalies reported by the algorithms.

There is a critical synergy between visualization and automatic detection from the perspective of the autonomic computing. Many traditional visualization tools are based on the organization of the system. In contrast, our tools present information in a format that is useful to operators, helping them monitor their system and allowing them to rapidly decide whether the visualization tool works and is helpful. From that foundation of trust, we then automatically point to behaviors at certain times that we think are suspicious using that same visualization format. The operators can then quickly decide whether or not these warnings are useful.

Without a visualization tool, we believe it would have taken scores of warnings for each operator to decide whether or not he or she trusted the tool. Since the detector and visualizer use the same metric, it makes it much easier and faster for the operator to decide whether the warning is a false positive. False positives thus become much *cheaper*; they are either significant behavior changes that operators may want to know about anyway, or they can be easily filtered out manually and visually. Since visual checking is quick, we may be able to afford a higher false positive rate in practice.

Acknowledgments

We would like to thank Mike Chen, Mark Verber, and Emre Kıcıman for their valuable insights and help with this paper. We also thank Alessandro Isolani, CEO of Ebates, for allowing us to use sanitized web log data.

References

[1] Altaworks, Altaworks Panorama. http://www.altaworks.-com/solutionsPanorama.htm.

[2] S. K. Card, J. D. Mackinlay, and B. Shneiderman. *Readings in Information Visualization: Using Vision To Think*. Morgan Kaumann, San Francisco, CA, 1999.

[3] M. Chen, E. Kıcıman, E. Fratkin, A. Fox, and E. Brewer. Pinpoint: Problem determination in large, dynamic internet services. DSN 2002.

[4] M. Y. Chen, A. Accardi, E. Kıcıman, D. Patterson, A. Fox, and E. A. Brewer. Path-based failure and evolution management. In *NSDI*, pages 309–322, 2004.

[5] I. Cohen, J. S. Chase, M. Goldszmidt, T. Kelly, and J. Symons. Correlating instrumentation data to system states: A building block for automated diagnosis and control. In *6th USENIX Symposium on Operating Systems Design and Implementation*, San Francisco, CA, Dec 2004.

[6] E. Eskin. Anomaly detection over noisy data using learned probability distributions. In *Proceedings of the International Conference on Machine Learning*, pages 255–262, 2000.

[7] Hewlett Packard Corporation, HP OpenView Software. http://www.openview.hp.com.

[8] IBM Corporation, IBM Tivoli Software. http://www.ibm.-com/software/tivoli.

[9] C. Kallepalli and J. Tian. Measuring and modeling usage and reliability for statistical web testing. *IEEE Transactions on Software Engineering*, 27:1023–1036, 2001.

[10] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *IEEE Computer*, 36(1):41–50, 2003.

[11] E. Kıcıman and A. Fox. Detecting application-level failures in component-based internet services. Technical report, Stanford, 2004.

[12] G. W. Snedecor and W. G. Cochran. Statistical methods. Eighth Edition, Iowa State University Press, 1989.

[13] J. Srivastava, R. Cooley, M. Deshpande, and P.-N. Tan. Web usage mining: Discovery and applications of usage patterns from web data. *SIGKDD Explorations*, 1:12–23, 2000.

[14] N. Ye, Q. Chen, S. M. Emran, and K. Noh. Chi-square statistical profiling for anomaly detection. In *IEEE Systems, Man, and Cybernetics Information Assurance and Security Workshop June 6-7, 2000 at West Point, New York*, pages 187–193, June 2000.