Failure Avoidance through Fault Prediction  
Based on Synthetic Transactions

Mohammed Shatnawi 1,2  
1 - Microsoft Online Ads Platform, 
Microsoft Corporation  
Redmond, Washington  
mohammed@microsoft.com

Matei Ripeanu 2  
2-Electrical and Computer Engineering Department,  
The University of British Columbia  
Vancouver BC, Canada  
matei@ece.ubc.ca

Abstract— System logs are an important tool in studying the conditions (e.g., environment misconfigurations, resource status, erroneous user input) that cause failures. However, production system logs are complex, verbose, and lack structural stability over time. These traits make them hard to use, and make solutions that rely on them susceptible to high maintenance costs. Additionally, logs record failures after they occur: by the time logs are investigated, users have already experienced the failures' consequences.

To detect the environment conditions that are correlated with failures without dealing with the complexities associated with processing production logs, and to prevent failure-causing conditions from occurring before the system goes live, this research suggests a three step methodology: i) using synthetic transactions, i.e., simplified workloads, in pre-production environments that emulate user behavior, ii) recording the result of executing these transactions in logs that are compact, simple to analyze, stable over time, and specifically tailored to the fault metrics of interest, and iii) mining these specialized logs to understand the conditions that correlate to failures. This allows system administrators to configure the system to prevent these conditions from happening.

We evaluate the effectiveness of this approach by replicating the behavior of a service used in production at Microsoft, and testing the ability to predict failures using a synthetic workload on a 650 million events production trace. The synthetic prediction system is able to predict 91% of real production failures using 50-fold fewer transactions and logs that are 10,000-fold more compact than their production counterparts.

Keywords—Failure prediction; failure avoidance; system logs; synthetic transactions; data analysis; data mining.

I. INTRODUCTION

Online services face an increasing demand for stringent Service Level Agreements (SLAs) that specify their availability, throughput, response time, or reliability. One way to improve service performance is to detect anomalous operating conditions that are correlated with SLA violations and attempt to mitigate these conditions [1].

To this end, the standard approach employed today is passive: service operators deploy the service for production use, analyze the generated production logs, and attempt to correlate the observed operational conditions with SLA violations (or service failures). Thus, they obtain information that can be later used to mitigate observed failure rates by controlling the service's environment. For example, after analyzing a month-long production log, a service operator might have statistically significant information that uncovers a correlation between service failures (e.g., SLA violations) and increasing machine load over the previous five minutes of running the service. In response, the operator may decide to introduce new provisioning policies that allocate more resources to the service when this trend is observed.

There are two inherent problems with this reactive operational mode. Firstly, by the time the failures are recorded, conditions analyzed, and problems fixed, the customers have already experienced the software problems and degraded service. This may cause irreversible financial or reputational damage to the enterprise. Secondly, production system logs are usually complex [7], often incomplete, i.e., the data in the logs may not always be sufficient for all data mining needs [5, 9], and preprocessing logs, to get them to a stage where they are usable for data mining is tedious and expensive [7, 9, 10].

While current research threads aim to address the above mentioned problems [1, 2, 5, 7, 9, 10], this research suggests a different, proactive approach. We advocate: using synthetic transactions in test environments, also known as pre-production environments, before the service goes live to identify the conditions that correlate with failures. We explore the use of compact, specialized logs that are tailored to specific SLA performance criteria (e.g., throughput
and response time). Finally, we demonstrate that the resulting logs can be mined to determine the system conditions that correlate with failures. Armed with this information, administrators can then configure production systems to prevent these conditions from happening. This approach overcomes two main problems: First, it exposes the correlations between system’s operating conditions and SLA violations ahead of production deployment, and does not wait until failures happen in production. Second, it is based on compact, uniform log files that reduce the scalability problems for analysis and mining.

We evaluate this approach in the context of an online transactional processing (OLTP) system used in production by Microsoft, and use the service SLA that specifies an upper bound for response time; i.e. if the time taken to execute the service is lower that what is specified in the SLA then the operation is successful; if the response time is higher, the operation failed.

For evaluation, we use both, logs obtained from executing synthetic traces, and logs from a large production service that contain about 650 million transactions. We show in (Section V) that by using synthetic transactions and specialized logs in pre-production environments, it is possible to project failure causing conditions with high accuracy (up to 91%) and an insignificant rate of false positives (about 0.1%). The time needed to reach such results is 50x lower than the time to generate production logs. Similarly, the synthetic logs are much more compact: they enable a 10,000-fold reduction compared to the size of production logs we have available.

Additionally, while exploring the effectiveness of the suggested approach, we also: compare the effectiveness and the overheads of two popular data-mining techniques that can be employed in this space (decision trees and Bayesian learning); and explore the tradeoffs between the volume of pre-production logs used to create and train the data-mining algorithms and their resulting accuracy.

We note that our approach is general in nature and can be used with any service, for example a Grid service, as a proactive means of correlating system’s usage, operating conditions, and resource allocation with the results these factors foster. Although we are limiting the discussion here to SLA violations as the class of failures we address, we believe that other failure types/classes can be effectively studied using the suggested approach.

II. BACKGROUND AND RELATED WORK

Before we proceed to present our proposed approach (Section III) and its evaluation, (Sections IV and V) we summarize the current threads of research that emphasize the problems of dealing with large and complex system logs, and the need to proactively address failures. The goal here is to provide a point of reference to the efforts leading to our work, and to highlight the difference between our approach and the other work in this area.

A. Proactive Failure Avoidance

A few research threads stress the need for proactive fault avoidance, as opposed to traditional reactive methods, to increase system’s reliability.

Pietrantuono et al. [1] argue that the ability to monitor a system at runtime and to be able to give estimations about its dependability trend is key to implementing strategies aiming at predicting and thus, proactively preventing system failures.

Cotroneo et al. [2] suggest the use of fault injection to make sure that no faults go undetected. They consider this a proactive approach and the goal is to improve production log effectiveness.

Our suggested approach is different in that it attempts to proactively understand the conditions that correlate to SLA violations, before deployment (i.e., in pre-production environments), and does not wait until failures occur at runtime to address them.

B. The Complexity of Handling Production-level Logs

Current research emphasizes the problems of building accurate data mining models based on production logs. First, the logs are complex and hard to handle and mine [7, 9]. Second, the data in the logs may not always be sufficient for all the data mining needs [5]. Finally, preprocessing the logs to get them to a stage where they are usable in prediction models is tedious and expensive [7, 9, 10]. The following is a brief literature survey that argues for complete, easy to use logs such as the ones we are suggesting.

Snyder et al. [5] argue that the insufficiency of log data causes problems for data mining them, because the data in the logs are extraneous, and it is hard to always realize and find which pieces of data in the logs are needed in the data mining process. They suggest aggregating data in the logs first before using it.

Zheng et al. [10] suggest event categorization and filtering of the logs to overcome the logs’ lack of structure and lack of amenability to data mining.

In their research, Xu et al. [7, 9] also explain the voluminous nature of data logs, and claim that logs are not actually helpful in a lot of cases, simply because of their large scale volume. They suggest log parsing and text mining to process the logs.

Whereas the research threads above attempt to overcome the problems and shortcomings of
production logs, our approach is different in that we create our own specialized, complete, simpler and compact logs that are amenable to data mining. The fact that we produce these logs before the system is deployed to production, not after the customers have endured the degraded service, is a second advantage.

### III. PROPOSED APPROACH

This section describes our suggested approach of using synthetic transactions, mining the resulting logs in predicting system failures, and correlating them with the operating conditions that might have caused them. The main motivating drivers behind this research are to overcome the problem of waiting for real, production logs to be collected, and to overcome the problem of dealing with production logs which are large, complex, incomplete, and change over time.

As depicted in Figure 1, our approach is to design synthetic transactions that emulate actual usage of the system before it is deployed in production. In doing so, we stress the system to produce failures (i.e., SLA violations), record the results (both success and failure) and the system parameters of interest into specialized logs, use the data in these logs to generate prediction models, use these to correlate the failures with the system conditions that caused them, and finally leverage this knowledge in configuring the system before deployment, so that these conditions do not happen in production.

The following subsections describe each of these steps in detail, while Figure 1 presents a summary of the complete workflow.

#### A. Synthetic Transactions Design

The first step, as depicted in Figure 1, is to understand the system’s functionality, and design a set of synthetic transactions to exercise it in a pre-production environment. There are various synthetic transaction designs; one design can be using synthetic workloads to exercise on the actual functionality of the system. For example, a customer of the live system may call a web service to create a user entity and register her own information in it to be used later to buy an online product. A synthetic transaction in the pre-production environment calls the same web service, creates a user entity and registers a random name and address for that user. Another synthetic transaction design can be to create simplified test operations that represent the main and critical functionality of the live system, and call those with synthetic workload. The choice of which synthetic transaction design to use should depend on the aspects of the system that need to be tested. If the system needs to be tested end-to-end, then the first design is more suitable. If a subset of the system needs to be tested, then the latter approach is more suitable. One common goal between all designs is full coverage of the system’s functionality of interest (i.e. performance, reliability, and/or availability).

The results of each synthetic transaction and the system state should be logged to be used in failure analysis. As a general guidance, we recommend designing highly specialized synthetic transactions, i.e. each synthetic transaction carries out one test and records information about one result. This results in logs that are more compact and in data that is strongly pertinent to the measure of interest.
performance of the function at hand, then performance related environment conditions like CPU load, memory pressure, and process information need to be logged.

Recoding all this data may result in large log files. One way to reduce the log file size is to take a multi-step approach and start by logging a wide set of parameters. Analyze those and eliminate the ones that have no or less impact on the measures of interest. This step could be repeated as many times as needed until a final, most relevant set of attributes are found. This is depicted in the first loop in Figure 1.

Last, but certainly not least, the log ‘schema’ (i.e., data attributes, types, and sizes) should be designed with data mining in mind, as the ultimate goal is to use it in predicting failures. Thus we recommend producing a log schema that matches the schema required by the data-mining tools that will be used.

Note that this section does not provide a particular log design; rather it is left to each experiment adopting this generic approach to decide the design of the logs. Later in Section IV.C, we show how this generic approach is used in designing a specific log schema.

C. Synthetic Transactions Execution and Logging

After designing the synthetic transactions and log schema, comes the part of exercising the transactions and logging their information. It is important to have a solid understanding of the usage pattern of the system, and emulate it with synthetic transactions. This requires understanding the traffic and workload patterns of the service in production. We strongly recommend stressing the system beyond its average projected usage by at least a factor of two. In our experience, this ensures enough headroom for unexpected behavior.

D. Mining the Logs

For the data mining process, we recommend using any of the commercially available data mining solutions. A data mining solution will create a data mining model, train it, and test it with the log data. This is depicted in the second loop in Figure 1.

The data mining step is made easier because of the investments in the synthetic transactions and specialized logs. There is no need for log pre-processing, and the resulting logs have the desired metadata and all the data pertinent to the mining requirements.

E. Pre-deployment System Configurations

The fifth and final step is to determine the actions to be taken before deployment, based on the mining results from the previous step. Here, administrators configure the system for production use and can make provisioning and/or admission control decisions to eliminate the conditions that have been determined to correlate with failures. For example, if by analyzing the pre-production logs it is determined that the service fails to adhere to its performance requirements when the number of active users passes a certain threshold, then the system administrators can use admission control mechanisms to limit the number of active users when the system is used in production.

IV. EXPERIMENTAL DESIGN

This section details the specifics of our experiment to better understand the feasibility and evaluate the benefits of synthetic transactions based approach.

We use a high-volume online transactional service deployed at Microsoft as our test case. We have intimate experience with the operational details of this service and we also have extensive production logs for the service. We design synthetic transactions following the methodology presented in Section III: generate a synthetic workload, exercise it on a pre-production replica of the system, and use data-mining to extract a predictor based on the resulting logs.

The following subsections present the details of executing the steps above for the online system we have chosen, and present the details of the hardware/software platform used. The next section (Section V) presents the evaluation results.

A. The Online System Used

We chose one of Microsoft’s online OLTP transactional services. The service provides entity create, read, update, delete and query functionality (CRUDQ operations). We also have access to five weeks of service’s production logs which average 129 million transactions a week. Each transaction log entry has an average size of 5KB, for a total of about 88GB a day.

The software and hardware we used to emulate usage of this service is:

- One SQL Server machine used as an OLTP server. This is a MS SQL Server 2008 standard edition, installed on Windows Server 2008 64-bit operating system, running on a machine with an Intel Core i7 920 (2.65GHz) CPU and 6GB RAM.
- One client machine that calls the synthetic transactions implemented on the OLTP server.
The synthetic transactions are implemented as SQL Server stored procedures. The exact specification of the client machine is not important for the experiment.

B. Synthetic Transactions

We studied the production OLTP systems’ transactions, and created a set of basic CRUDQ operations that emulate the services’ operations. For example, we created an operation that creates a user entity, populated it with random data for name and address. We did the same thing for the rest of the CRUDQ operations. These synthetic transactions were implemented on the OLTP server, and were called from the client machine in a way that emulates users’ requests over a period of 8 hours.

The running period emulated peak usage for 25% of the time (2 hours), slow usage for 25% of the time, and average usage for 50% of the time. This workload distribution was made based on the actual OLTP design considerations. The CRUDQ operations we created include the basic operations with sanity checks to emulate, as much as possible, the operations of the service. So between the similarity of the service’s functionality and load distribution, we believe the synthetic workload is close to the real one.

The measure of interest in this experiment was response time SLA.

C. Log File Design

In order to define a compact log that records the most impactful factors on response time, we started by logging all the information we thought would affect it. This included transaction name, type, time, response time, number of processes at the time, CPU utilization at the time, number of users logged in, and memory usage. We iteratively refined the set of data that was logged, until the following attributes were found to be the most impactful in determining operations’ response time:

- **Transaction Type** – This is the synthetic transaction that was executed (create, read, update, delete, or query). Obviously, query was the most expensive, followed by update, create, delete, and finally read.
- **Number of concurrent processes** – the total number of active processes at the time of measurement.
- **Time** – The time the test took place. This allows for finding the chronological trace of events, and correlating results to the factors that caused them (i.e. failure happened at 12:00:00.000pm, and in the past five seconds, CPU utilization was increasing)
- **CPU Load** – This is the percentage of CPU usage when the transaction took place.
- **CPU Load - Recent History Trend** – This is a measure over CPU usage which shows the past few seconds of CPU utilization (i.e. has it been going up, steady, going down.) This is not recorded in the log, but rather calculated from the data mart.

Using dimensional model terms defined in [8], the above attributes are the dimensions. In other words, these are the factors that affected response time. The measures of interest for this experiment are:

- **Response time** – the time it took the transaction to execute. This is the success criteria specified in the service’s SLA.
- **Test result** (success/failure) – an operation is considered successful if it is executed within the expected response time specified in an SLA.
- **Average Response Time for the past 5 seconds** – This is an aggregation step, i.e. not recorded in the log, rather it was calculated from the data mart to evaluate the CPU recent history trend. We calculated this value as part of our experimentation, as we noted that most failures happen after a period of high CPU load. So as part of our approach of refining the data to be used in data mining, we attempted a few measures, and found this one to be impactful.

D. Log Data

The synthetic transactions discussed above, were run in the pre-production environment and produced data of the schema designed in the previous section.

The synthetic batch had about 28 thousand CRUDQ transactions. Each logged record was about (400 Bytes) in size, which resulted in a file size of almost 11MB for the duration of the 8 hours.

The synthetic data log needed no further analysis to prepare it for data mining, as it was logged into a schema designed with data mining in mind.

E. Mining the Logs

So far, we have designed the synthetic transactions, the log schema, and generated the data to be used in the mining process. To perform the actual data mining process, we used the same SQL Server machine described above, and the following two software solutions:
Microsoft SQL Server Integration Service 2008 (SSIS) ETL application was used for extracting data from the logs, cleaning it, and populating it into the data marts. It was also used in pre-aggregation steps to calculate the intermediate results (Average response times, and CPU historical trends).

Microsoft SQL Server Analysis Services 2008 (SSAS) was used in data mining. Two data mining algorithms were used: decision trees and naïve Bayes.

We chose these two mining algorithms because of their effectiveness in predicting discrete values (i.e., success and failure), and because they complement each other in terms of prediction ability. The decision trees algorithm is good at showing a hierarchy (a tree) of the factors that affected the result and how the factors relate to each other. The Bayesian algorithm is good at predicting discrete values.

**Training Data Set** – to train the data mining model, to allow it to learn the data patterns that correlate to the various results, we choose 20% of the synthetic data as a training set. We chose the training set to be evenly distributed between successful and failed transactions. For more details on how to choose the training set, and why not use all the data set for training, please refer to [4] and [11].

**Testing Data Set** – we used 30% of the synthetic transactions as the testing set. We got an average of 99% prediction accuracy using the testing data. If poor testing results were found, the training set would need changing to ensure good coverage of the various data patterns correlating the inputs to the results.

**Data Set** – we used the remaining 50% of the synthetic log data as the data set (source data) for the actual prediction of the conditions that correlate to system failures.

It took us three attempts of training, with each of them taking about 30 minutes of data selection and mining. This was because we had used only 20% of the synthetic data set as a training set.

The following section evaluates the prediction ability of the trained predictor obtained with the methodology presented so far.

**V. RESULTS**

This section presents the results of evaluating our proposed methodology. To verify the accuracy of the prediction models extracted in pre-production environments using synthetic transactions, we used the failure forecasting models that resulted from mining the synthetic logs and compared their predictions with the actual failures and operating conditions observed in the production logs (sections V.A and V.B).

As an added validation step, we mined the production logs themselves, created a failure predictor, and compared the performance of this predictor with that obtained from synthetic logs (Section V.C). This validation step enables a direct comparison between the accuracy of the synthetic based mining system and the production based one in terms of predictor accuracy, and, equally importantly, in terms of computational and storage effort to mine the logs.

### A. Failure Prediction Based on Bayesian Learning

We applied the Naïve Bayes algorithm on the logs produced from synthetic transactions, and the prediction results showed that “CPU Load” and “Recent Average Response Time” played the most impactful roles in causing faults. The Bayesian learning algorithm found that if the CPU load was higher than 77%, and the recent CPU response time was increasing during the past 5 seconds before the time of the current transaction, then the current transaction will fail.

We validated the accuracy of these insights by analyzing the production logs. Figure 2 compares, for each week, the actual faults recorded in the real logs with the ones a predictor based on the Bayesian logic, and trained using synthetic transactions, forecasted. Out of an average of 129 million weekly transactions, the average failure rate observed in the production trace was 1505 faults a week. The Bayesian system predicted 1374 of those (a prediction accuracy of 91.27%) and had a low rate of false positives (on average one failure a week, less than 0.1%).

![Figure 2 – Number of actual faults and predicted faults using naive Bayes algorithm](image)
We stress that we used only one day of synthetic transactions for the traces that are used to train the predictor. Later in this section we show the results found from using a prediction system generated from real production logs.

**B. Failure Prediction Based on Decision Trees**

We, also, applied the decision trees learning algorithm on the logs produced from synthetic transactions, and the prediction results showed that “CPU Load” predicts best the occurrence of failures. The decision trees learning algorithm emphasized CPU load buckets. It determined that for CPU load greater than 77%, all operations will fail, and it had detailed considerations for CPU usage down to 68%.

Here too, we validated, using the production trace, the prediction based on decision trees. Figure 3 compares, for each week, the actual faults recorded in the production logs with the ones a predictor based on the decision trees logic, and trained using synthetic transactions, forecasted. Out of an average of 129 million weekly transactions, the average failure rate observed in the trace is 1505 faults a week. The system predicted 1333 of those (a prediction accuracy of 88.58%) and still a low rate of false positives (on average 5 failures a week, or about 0.3%).

These results show that the Naïve Bayes algorithm is more accurate than decision trees in predicting faults for this experiment.

**C. Using Production Logs to Inform Learning**

To further validate the results of our experiment, and more importantly, to put into perspective the savings enabled by our solution based on synthetic logs, we evaluated the predictor accuracy when created and trained using real production logs. We followed a similar procedure: we built data mining models and structures based on the production data, split the real logs into training, testing and data sets, ran the learning algorithm on the training set, and evaluated the accuracy of the predictor based on the rest of the data.

The production based predictors can produce up to 100% prediction accuracy, yet this comes at a high cost in two respects. First, handling large log files, in the order of hundreds of Gigabytes per day. The effort to pre-process and manage this volume of data is about 12 hours of software execution per day (for log extraction, ETL, and analysis). Second, the data cleansing and analysis effort on the original production log managed to reduce the log file size to about 10 GB per day. This is still a large file. To keep the data mining cost within reason, i.e., within a couple of hours of training and testing, we had to further group the production data into buckets of common factors, eliminate some of the factors that are less impactful on execution results, and reduce the data set into about 1GB of data. This is a considerable effort, and took about 4 hours of ETL processing on the data set. Note that, in addition to these costs, users had to actually endure the SLA violations.

In contrast, our solution using synthetic transactions saves users from enduring the degraded service, results in a much smaller and complete log files, and saves a lot of time and effort in creating and using the prediction system. The results of our synthetic based prediction system (high prediction accuracy, low rate of false positives) are satisfactory and comparable to the production predictors’ results.

**VI. DISCUSSION**

The main complexity involved with the proposed approach in general, and with the experiment we describe in particular, is understanding the service at hand, reproducing the service in pre-production environments, and generating accurate synthetic transactions to emulate actual service usage. There are ways to deal with this complexity but they still impose a challenge to system operators.

Understanding the service, the quality measures of interest, and the quality of service requirements is difficult. The ability to isolate the measures of interest and focus on them within an intricately interconnected system is difficult, and may not always be attainable. This is also true for the log design. In cases where it is not possible to isolate one measure of interest in synthetic transaction and log designs, the experiment designers may group measures of interest in their tests and logs.
Reproducing the system in pre-production environments can be simplified if the whole production system is replicated; and this is not uncommon in large enterprises where the critical-path sub-systems are replicated and thoroughly tested before production. In cases, where this is not feasible, our approach is still applicable, using emulation techniques.

As far as the synthetic transactions being a true representation of the system, this will depend on the thorough understanding of the service, its API calls, and it is usage patterns. The challenge is exacerbated by the complexity of the service, and its inter-system dependencies. We believe historical information about and studies of the system’s real usage may help in the design of better and more relevant synthetic transactions.

VII. SUMMARY AND FUTURE WORK

This paper proposes an approach to enhance an online service’s reliability, availability and performance. We suggested using synthetic transactions in pre-production environments, and combined with data-mining on compact, specialized logs for predicting the failures in the system, and the environment conditions that that correlate to failures.

The advantages to this approach are manifold. The analysis and predictor training occur before the system goes to production. This enables detecting in advance the environment conditions that lead to failures and possibly taking action in advance thus increasing service reliability. A second, equally important, advantage is that the data set used in creating and using the synthetic predictor systems are orders of magnitude smaller, easier to use, and faster to process than their production counter parts.

As a future follow up, it would be interesting to see the impact of utilizing synthetic transactions in live systems, not only in pre-production environments, and to combine their effects with real time data analysis, not only pre-production data analysis and historical data.

One immediate follow up to this paper, is to study the effect of increasing the size of the synthetic data set, and experimenting with other data mining algorithms to increase the prediction accuracy.

REFERENCES