Retrieving Relevant Reports from a Customer Engagement Repository

Dharmesh Thakkar, Zhen Ming Jiang, Ahmed E. Hassan
Software Analysis and Intelligence Lab (SAIL)
Queen's University, Kingston, ON, Canada
{thakkar, zmjiang, ahmed}@cs.queensu.ca

Gilbert Hamann, Parminder Flora
Research In Motion (RIM)
Waterloo, ON, Canada

Abstract

Customers of modern enterprise applications commonly engage the vendor of the application for on-site troubleshooting and fine tuning of large deployments. The results of these engagements are documented in customer engagement reports. The reports contain valuable information about the observed symptoms, identified problems, attempted workarounds and the final solution. Such information is valuable in supporting analysts in future engagements. Engagement reports are stored in a customer engagement repository. Retrieving relevant reports from such a repository is usually ad-hoc and is based on using basic text search.

We present a technique to retrieve relevant reports from an engagement repository. The technique takes as input an execution log for a particular deployment and retrieves relevant engagement reports. The technique identifies relevant reports by comparing execution logs attached to the report stored in the engagement repository. The technique returns two types of relevant reports: (1) reports for engagement with similar operational profiles to identify prior engagements with similar workloads and (2) reports for engagement with similar signature profiles to identify prior engagements with similar problems. Using our technique, support analysts can locate relevant engagement reports and use the knowledge in them to quickly resolve problems at hand. To demonstrate the feasibility of our technique, we present two case studies: one case study uses an industry standard open source application, the Dell DVD Store, while the other case study uses a large enterprise application. The results of our case study show that our technique performs well in identifying relevant reports with high precision and high recall.

1. Introduction

Modern enterprise applications are used by a large and varied customer base. Support analysts must interface with hundreds or thousands of customers for troubleshooting and fine tuning their deployments. Techniques are needed to help the support analysts by making use of their acquired knowledge from prior support engagements.

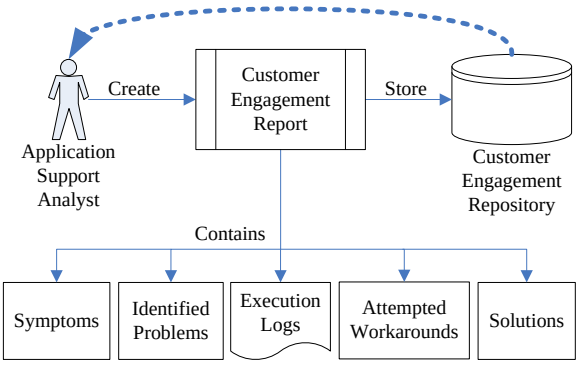

Figure 1: Customer Engagement Reporting

Support analysts often use their experience to troubleshoot the application deployment according to the operation profile, problem symptoms, operating system and hardware platform of the deployment at the client site. As shown in Figure 1, at the end of a customer engagement, the application support analyst creates a *customer engagement report*, which captures observed symptoms, identified problems, attempted workarounds and the final solution. One or more execution log files from the customer site are attached to the report. At mature software development organizations, these customer engagement reports are archived in a Customer Engagement Repository. The repository contains practically acquired invaluable information, which can be useful in many ways [1]. However, retrieval of information from this repository is not well explored. There exist no systematic techniques to retrieve and use information in such a knowledge base for future engagements. In this paper we present a technique to support future engagements by reusing information stored in the customer engagement repository. This idea is represented by the dotted line connecting the repository to the analyst in Figure 1.

When working on a particular engagement, support analysts rely on their experience in identifying prior engagements with similar circumstances. Analysts commonly use basic text search technology to retrieve relevant reports of prior engagements using specific keywords. However, such an approach requires

consistent entry of the data in the reports and the use of the appropriate keywords in the search. For example, a search for "hung thread" would not return a report which talks about a "non-responsive thread". The use of basic search technology all too often prevents the analyst from quickly locating the appropriate reports.

In this paper we present a technique which uses the execution logs attached to the engagement reports to help retrieve relevant customer engagement reports from the engagement repository. The log files provide certain advantages over other pieces of information attached to report (see Figure 1). Execution logs are consistent since they are automatically generated by the application, while all other information in the report is manually entered. The manually entered information might suffer from issues like, (a) varying level of completeness of information (b) inconsistent use of terms (b) analysts' incomplete knowledge of all operations and features of the application (c) analysts' inexperience, leading to bias towards documenting known territories. On the other hand, the execution logs are a direct representative of the application's operations and problems.

Our technique takes as input an execution log file for a particular deployment and returns relevant engagement reports. The technique returns two types of relevant reports:

(1) **Reports of engagement with a similar operational profile.** The operational profile identifies the workload characteristics of a particular application deployment. For example, given a deployment of an email server with an operational profile where 80% of the traffic is outgoing email and 20% in incoming email, our technique would return engagement reports for deployments with similar profiles. These reports are valuable when investigating workload problems (e.g. slow response time under a particular workload).

(2) **Reports of engagement with a similar signature profile.** Whereas an operational profile summarizes the workload characteristics of an application, a signature profile identifies the characteristics which are most peculiar for a particular deployment relative to all other deployments. For example, if a deployment has a relatively high number of deadlock events, then our technique would return engagement reports for deployments with relatively high number of deadlock events, even though that deployment might be similar to some other deployment with respect to its workload characteristics. These reports are valuable when investigating configuration and environment problems (e.g., environment error messages, hung threads, and restarts).

Our technique uses readily available yet hardly used information in the customer engagement repository. Analysts can pick the type of retrieval method to use depending on the situation at hand. For example, if a deployment is facing a problem of higher response time in some transactions, it would be appropriate to retrieve reports based on the operational profile. If a deployment is facing isolated occurrences of applications restarts or hung threads, it might be appropriate to retrieve relevant reports based on the signature profile. We show the validity and usability of our technique in practice through case studies performed on two applications – first the Dell DVD Store application, and second a large enterprise application. Our results confirm the high performance (i.e., precision and recall) of our technique.

2. Related Work

Due to space constraint, we will focus on works related to mining customer engagement repository. However, we believe the technique itself is independent of engagement reports: for instance, it could be used for retrieving related bug reports based on logs attached to bug reports. Related areas of research also include path profiling and fault diagnosis techniques.

The work most closely related to our work is by Hui and Jha [1] who mine the customer engagement repository in the manufacturing industry to retrieve relevant reports. The authors apply machine learning techniques on the textual service records stored in the repository. In contrast, because of the inherent limitations in entering and searching textual information noted earlier, our work is based on applying statistical techniques on the execution logs attached to the engagement reports.

We use execution logs as representatives of operational profiles which capture the workload characteristics of an application. There are many techniques to create operational profiles, such as [2, 5 and 12]. Our work is different from the aforementioned approaches in that it goes on to compare execution logs based on the operational profiles that they represent, without actually retrieving operational profiles.

Unlike operational profiles, signature profiles are not well explored by researchers and practitioners. Researchers working in areas related to software quality and reliability often analyze signature events in an application [3, 15]. However, retrieval of engagement reports based on the signature profile, as done in this work, has not been proposed.

3. Structure of Execution Logs

An operational feature of an application is made up of one or more code modules. A code module can generate

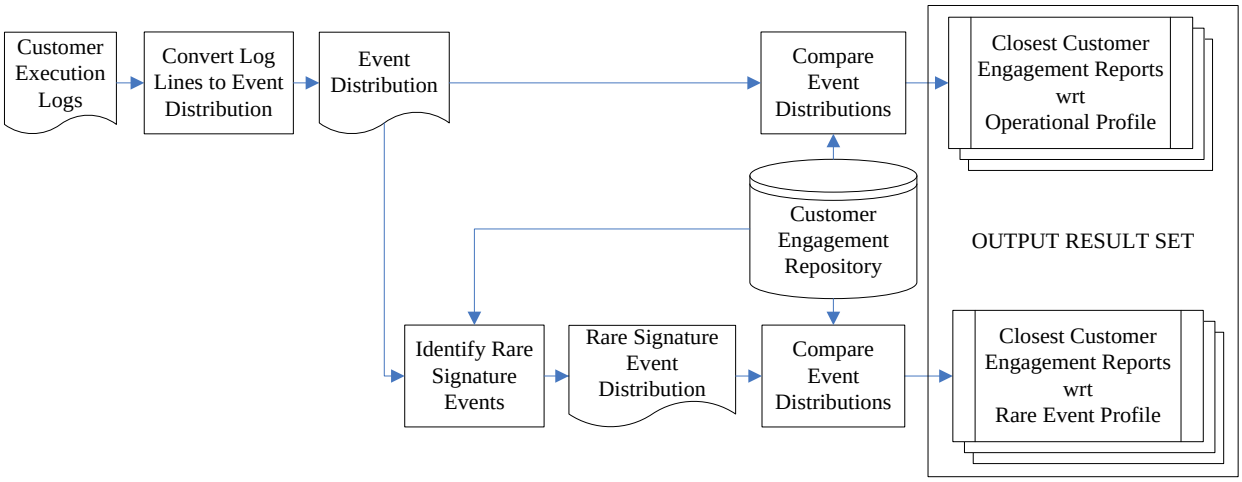

Figure 2: Our Technique to Retrieve Relevant Engagement Reports

one or more events in the execution log once it is executed. Thus, one of the most readily available information related to application usage at any customer deployment site is the execution logs (or activity logs). The execution logs typically contain time-stamped sequence of events at run-time. Table 1 shows a sample execution log for an enterprise collaboration suite, such as Zimbra [10] or Microsoft Exchange Server [11]. Note that execution logs tend to be quite large in size, as they record code module level activities at runtime.

Execution logs help remote debugging by providing a detailed context for field issues. While many applications are designed with their own logging mechanisms, logging frameworks such as the Apache Logging Services [4] can be used to enable event-logging in applications.

Table 1: An Example of Execution Logs

<time> Queuing new mail msgid=ABC threadid=XYZ
<time> Instant msg. Sending packet to client msgid=ABC threadid=XYZ
<time> New meeting request msgid=ABC threadid=XYZ
<time> Client established IMAP session emailid=ABC threadid=XYZ
<time> Client disconnected. Cannot deliver msgid=ABC threadid=XYZ
<time> New contact in address book emailid=ABC threadid=XYZ
<time> User initiated appointment deletion emailid=ABC threadid=XYZ

Legal Requirements on Application Logging

In response to increased accounting and security regulations, governments in various nations created laws requiring the logging of the execution of enterprise and financial applications. For instance, the Sarbanes-Oxley Act of 2002 [16] in the US, and the EU directive of 2006 on data retention [13] are major steps in that direction. These legal requirements helped increase the availability of the execution logs required as an input for our technique. There has also been an increased concern over privacy and security information present in the execution logs. It is common to remove such sensitive information from the logs before passing it for application analysis. Our log mining technique works equally well on such anonymized execution logs.

Execution Logs vs. Tracing Logs

Execution logs are routinely generated at customer installation site according to selected logging levels. Execution logs contain activity events (such as "Account verified" or "Message delivered") as well as error events (such as "Message queue full" or "Too many requests, server busy"). In contrast, tracing logs (or implementation logs) are generated by code instrumentation or statistical sampling using profiling tools. Tracing logs provide lower level details, such as logging of each function call during runtime (such as "Function CheckPassword() called").

While tracing logs provide more accurate details, they are mainly used during development [6] and result in a high overhead on the application. Hence tracing logs are not normally available at customer sites, though they might provide a better representation of the operational and signature profiles. For this work, we use the readily available execution logs.

4. Our Technique for finding Related Reports

Our technique takes the execution log files for a customer as an input and retrieves the relevant execution log files and corresponding customer engagement reports from the engagement repository. The execution log retrieval is based on (a) similar operational profile, and (b) similar signature profile. Figure 2 summarizes the steps of our technique. Each step in the Figure is discussed in the following subsections.

4.1 Convert Log Lines to Event Distribution

Execution logs are composed of dynamic and static information. Each log line has static information about the execution event and dynamic information that is

Table 2: Example Event Distribution

ID	Event	Original Event Distributions			Operational Event Distributions			Signature Event Distributions		
		R1	R2	R3	P1	P2	P3	S1	S2	S3
E1	New Message	4000	3500	1000	44.39	46.66	22.16	0.02	0.02	0.05
E2	New Contact	3500	3000	1500	38.84	39.99	33.24	0.03	0.03	0.03
E3	New Meeting Request	1500	1000	2000	16.64	13.33	44.33	0.06	0.08	0.03
E4	Message Queue Full	5	1	6	0.06	0.01	0.13	18.02	75.01	7.52
E5	Connection Lost	7	0	6	0.08	0.00	0.13	12.87	0.02	7.52
Total		9012	7501	4512	100	100	100	31.01	75.13	15.14

specific to the particular occurrence of that event. We must convert execution log lines to log events, so we can recognize if two log lines correspond to the same event even if they are processing different data. We use an approach which employs clone detection techniques to identify for each log line the variation points relative to their log lines [17]. For example, given the two log lines "Open inbox user=A" and "Open inbox user=B", our technique would map both lines to the event "Open inbox user=?".

Once the log lines are abstracted to events, we obtain a distribution of log events by event counting. The event distribution is then normalized as the percentage of each event in the event log, so that we can compare event logs for different running times without bias. For retrieval based on signature profile, we want to give higher weight to the events occurring at lower than normal rate (rare events) over events occurring at higher than normal rate. To give events with lower than average occurrence a boost in the distribution, the frequency for each event is inverted in the signature distribution.

An example of three logs files is shown in Table 2. R1, R2 and R3 represent the original event distributions of three log files F1, F2 and F3 respectively. Their corresponding operational event distributions P1, P2, and P3 are used for operational profile based retrieval. The last three columns S1, S2 and S3 show the inverted event distributions used for signature profile based retrieval. This example is used as a running example in this section. Looking at the frequencies of the events in R1, R2 and R3, we expect our technique to show that F1 is closer to F2 in terms of operational profile, while F1 is closer to F3 in terms of signature profile. Note that this is a very small example intended to show our technique at work. For real applications, the number of events is expected to run into hundreds or thousands, instead of just five events as considered in this example.

4.2 Identify Signature Events

A signature event is a rare, i.e., infrequent event in a log file relative to the occurrences of all events in other log files from other deployment stored in the repository. Events such as dropped connections, thread dumps, and full queues are examples of signature events. Instead of searching for such events in log files in a hard-coded way, we examine the distribution of the events in all log files and we pick the events that are occurring at rates that vary considerably from the norm. These signature events indicate potential problems or outlier execution paths that are experienced by the application. We need to create a signature event distribution which is a subset of the operational event distribution, as it contains only the signature events.

Table 3: Contingency Table for Chi-Square Test

	Frequency of Event (E)	Frequency of Other Events
In input log file	18.02	12.99
In all other log files	41.26	3.87

We use a *chi-square test* with a p-value of 0.1 to identify events occurring at frequencies that are statistically different (higher or lower) than their usual occurrence frequency. The chi-square test determines if for the input file whether a particular event (E) is occurring at a frequency that is consistent with occurrence of that event (E) in the rest of the log files in the repository or not. Table 3 shows a contingency table for the event E4 in the log file F1 from the example considered in Table 2. The chi-square test flags that event E4 is occurring at a different rate in the input file F1 than usual. For the events in Table 2, the chi-square test flags events E4 and E5 as signature events for S1; E4 for S2; and E4 and E5 for S3. Only these events are considered part of the signature event distribution for each log file. The events filtered out by chi-square test are grayed out in the columns S1, S2, and S3 in Table 2.

4.3 Compare Event Distributions

After the previous two steps, we have an operational event distribution and a signature event distribution for the input log file and all the log files in the repository. The characteristics of the event distributions vary depending on the operational profile and problem symptoms of a customer. If two customers have similar

operational profiles, they would have similar event distributions. Figure 3 shows three different distributions of events for visual examination. The horizontal axis represents different events in the event distributions and the vertical axis represents the frequencies of those events. Visual inspection reveals that distributions D1 and D2 are similar to each other, compared to D1 and D3, or D2 and D3. We measure the distance between event distributions using two metrics: the Kullback-Leibler divergence and the cosine distance.

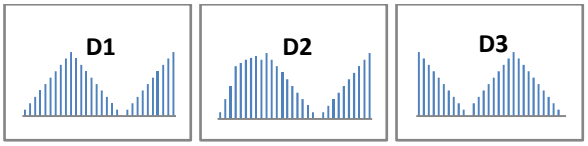

Figure 3: Visually examining distributions similarity

In the next two subsections, we discuss the distance metrics that we used to measure the similarity between two event distributions. We also apply these metrics on the example presented in Table 2. For a given input log file, our technique calculates the distance from all the log files in the repository, to find the most relevant. Our algorithm is summarized as follows:

For the input execution log
Obtain operational events distribution P
Obtain signature events distribution S
End For
For each execution log in the engagement repository
Obtain operational events distribution P_i
Obtain signature events distribution S_i
Calculate operational distance between P and P_i
Calculate signature distance between S and S_i
End For
Sort and present engagement reports by ascending order of operational and signature distances

4.3.1 Kullback-Leibler Divergence Metric

Given two distributions P and Q, the Kullback-Leibler divergence [8] (here after called K-L divergence) between P and Q is defined as:

$$KL(P,Q) = \sum_x P(x)\, log \frac{P(x)}{Q(x)}$$

K-L divergence is sometimes referred to as relative entropy or information gain. K-L divergence is not a distance metric in the strictest sense, because it is not symmetric, and the triangle inequality does not hold. That is, KL(P,R) is not equal to KL(R,P), and KL(P,R) can be greater than KL(P,Q)+KL(Q,R). To surmount the asymmetry limitation, we define distance $D_{KL}(P,Q)$ as the sum of KL(P,Q) and KL(Q,P). The smaller the D_{KL} value, the closer the distributions are.

We now show the use of K-L divergence using the example of Table 2. For this example, the operational profile distance $D_{KL}(P1,P2)$ is 0.41, $D_{KL}(P1,P3)$ is 18.9, which confirms that the operational profiles for F1 and F2 are closer compared to F1 and F3. The signature profile distance $D_{KL}(S1,S2)$ is 35.29, and $D_{KL}(S1,S3)$ is 5.24, which confirms that signature profiles for F1 and F3 are closer compared to F1 and F2.

4.3.2 Cosine Distance Metric

To compare two event distributions, they can be represented as vectors and similarity can be drawn in terms of the geometric distance. Each event type can be considered as a dimension and the frequency of occurrence of an event type can be considered as the weight in that dimension. Thus, for a given event log, the relevant event log is the one with the minimum distance in the vector space. One widely used distance metric in this context is the cosine distance, which is defined as:

$$D_C(P,Q) = \frac{\sum_x P(x)Q(x)}{\sqrt{\sum_x P(x)^2 Q(x)^2}}$$

In information retrieval systems, the cosine distance has been used as a similarity measure between two vectors representing two entities, such as queries, documents or web pages. If the two vectors are similar (congruent in geometric representation), the cosine distance reaches its maximum value, 1. If the vectors have least in common (perpendicular to each other in the geometric representation), the cosine distance reaches its minimum value, 0.

For the example in Table 2, the cosine distance for operational profiles $D_C(P1,P2)$ is 0.998 and $D_C(P1,P3)$ is 0.824, which quantify that F1 is closer to F2, compared to F3. The cosine distance for signature profiles $D_C(S1,S2)$ is 0.814 and $D_C(S1,S3)$ is 0.986, which confirms that signature profile for F1 is closer to F3, compared to F2. In this example, the results for both K-L and cosine distance metrics are consistent, but in practice the results may vary. We explore both distance metrics in our case studies in Section 5.

4.4 Measuring Performance

To measure the performance of our technique, we employ traditional metrics for information retrieval: precision and recall [7]. Our technique retrieves the most relevant log files for a given execution log file. For example, if the set of relevant log files for a given a log file F is C = {F1, F2, F3} and our technique returned the set R = {F1, F3, F4, F5}, as shown in Figure 4, then we measure precision and recall as follows:

CR = {F1, F3} is the intersection of the sets C and R. For our example, the precision would be 2/4 = 50% and the recall would be 2/3 = 66%. An optimal retrieval technique is the one which produces the best values for both precision and recall.

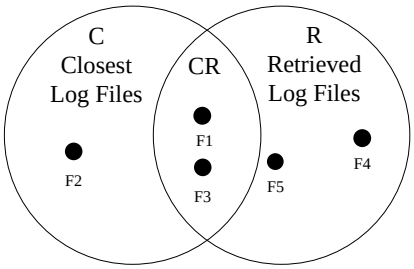

Figure 4: Precision and Recall

$$Precision = \frac{CR}{R} \qquad Recall = \frac{CR}{C}$$

The precision and recall measures above are applicable to a single log file. To measure the accuracy of our technique over several log files, we use the average precision and recall as follows:

$$Average\ Precision = \frac{1}{N} * \sum_{i=1}^{N} Precision_i$$

$$Average\ Recall = \frac{1}{N} * \sum_{i=1}^{N} Recall_i$$

Here N is the total number of log files on which we applied our technique.

5. Case Study

To study the effectiveness of our technique, we conducted a case study using synthetic and field deployment logs for a test application and an enterprise application. Using the synthetic logs we could measure the performance of our technique under specific simulated settings. Using the field deployment logs, we can measure the performance of our technique in a real life setting. We present the two case studies in the following subsections.

5.1 Case Study on a Test Application

The application

Our first test application is the Dell DVD Store application. The DVD Store (DVD Store 2 or DS2) application is an open source enterprise software application. The DS2 application is developed by Dell as a benchmarking workload for white papers and demonstrations of Dell's hardware solutions [14]. DS2 seeks to emulate today's online stores, architecturally and functionally. DS2 has a three-tier architecture. DS2 includes application server components, database server components and a load generator engine (client emulator).

The DS2 load generator emulates website users by sending HTTP requests to the application front-end. The DS2 application front-end encodes the various business rules, such as ordering new titles, declining an order in case of insufficient inventory. All customers, titles and transactional data are stored in the database server tier. We chose DS2 over other applications since it is an open source application which others can download to easily compare our results with their work.

Table 4: List of Log Files Tested for Retrieval

	New Customer Percentage	Average Number of Searches per Order	Average Number of Items Returned in each Search	Average Number of Items per Order
F1	100	5	5	5
F2	50	5	5	5
F3	20	5	5	5
F4	0	5	5	5
F5	20	1	5	5
F6	20	10	5	5
F7	20	5	1	5
F8	20	5	10	5
F9	20	5	5	1
F10	20	5	5	10

Experiment Setup

To demonstrate the feasibility of our technique, we need to create a repository of log files. We generate a large number of log files based on various simulated runs of the application. For each log file we ensure that we produce other relevant log files. Once we have these log files, we can pick each log file and use our technique to retrieve other relevant log files. We can then measure the precision and recall of our technique.

Since the DS2 application is a benchmarking application that is not intended for production deployment, it is not designed to generate execution logs. So for the purpose of our case study, we instrumented the application code to generate logs for execution events. The main operational features in the DS2 application are: Create Account, Login, Search the Store, Add to Cart and Checkout. We instrumented the code so that each of these operational features would generate a balanced number of execution events (four events each). So we have 20 different execution events in total.

We applied different synthetic workloads and collected the resulting execution log files. Table 3 lists the log files we collected. Each column in the table is a workload parameter, whose unique value makes operational features exercised in corresponding proportion, leading to unique execution logs. New Customer Percentage parameter identifies percent of the order cycles that would exercise the Create Account feature. Its value can be between 0 and 100, its typical

Table 5: Performance of Retrieval using Operational Profiles

Experiment	Count of Log Files	K-L Distance		Cosine Distance	
		Precision	Recall	Precision	Recall
Single Feature Group	28	67.71%	90.28%	67.71%	90.28%
Multiple Feature Groups	28	60.71%	80.95%	75.00%	100%
All Feature Groups	12	72.92%	97.22%	62.50%	83.33%
Real World Log Files	12	54.17%	72.22%	68.75%	91.67%
All the Log Files	80	59.93%	79.90%	56.72%	75.62%

value is 20. Average Number of Searches per Order identifies the number of times the Search operation should be performed in an order cycle, its typical value is 5 in the application. The remaining two parameters Average Number of Items Returned in each Search and Average Number of Items per Order are fairly self-explanatory. To create our repository of logs, we conduct load sessions with three different settings for each of the parameters – the typical value for the parameter and a value on either side of the typical value – for each of these parameters, while keeping the other parameters at their typical value. The resulting list of operational profiles is presented in Table 4, in which each row is a unique log file.

Since we aim to retrieve relevant log files with similar operational profile, we reran all the load sessions of Table 4 three times each. Thus, for each log file our technique for similar operational profile should return the three log files corresponding for the three reruns.

To test the performance of our technique for signature profile based retrieval, we need log files with similar signature events, as well as log files with different signature events. We obtained log files with signature events by introducing known problems in the DS2 application code, and sporadically invoking those problem paths in a controlled fashion using the load generator. For instance, we changed the application code to submit an ill-formatted SQL command to the database if a purchase order has more than 25 items, resulting in an exception event in the execution log. To sporadically invoke this problem path, we configure the load generator to create less than 0.5% of all the purchase orders with more than 25 items.

We introduce same sporadic problem events in operationally different log files listed in the Table 4. That is, similar problem events are added to groups of log files as {F1, F2, F3, F4}, {F2, F3, F4, F5}, {F3, F4, F5, F6}, and likewise. Hence the expected retrieval results from the technique are the log files having similar signature events, irrespective of the similarity in the operational events. That is, the expected result set for F3 are F1, F2, F4, F5; expected result set for F4 are F2, F3, F5, F6; and likewise.

Experiment Results

Using our technique, we could correctly retrieve the relevant operational profile and relevant signature profile with 100% precision and 100% recall using both the K-L divergence and cosine distance metrics.

5.2 Case Study on an Industrial Application

Our second application is a multithreaded enterprise application deployed at many organizations worldwide. The application provides some of the common enterprise communication features, such as email and calendar synchronization. With more than 700 unique execution events (compare to 20 unique execution events in DS2 application), we believe the application is a complex real-life enterprise application.

5.2.1 Studying Retrieval by Operational Profile

We studied the effectiveness of our technique on many different experiments. In the next subsections, we present these experiments. Although these experiments do not cover all possible real world operational profile comparison situations, we believe they represent the breadth of it. Table 5 summarizes results of all the experiments. We discuss each of the experiments in the following subsections.

5.2.1.1 Single Feature Group

In this experiment, we use log files of workloads with a single feature group of the application. A feature group is a set of related operational features of the application. For example, a enterprise collaboration suite such as Zimbra or Microsoft Exchange Server has feature groups like emails, instant messages, calendar, and address book. A feature group has features, for instance, email feature group has operational features send email, receive email and delete email. In this experiment, each execution log file is obtained by exercising a different feature group of the application using a workload generator. We exercised seven individual feature groups of the application, providing log files that represent seven different operational profiles. Then we rerun each of the seven workloads three times each, to obtain log files which represent similar operational profiles. Thus

for each log file, our technique is expected to return the three log files for the three reruns. We have a total of 28 log files, for each which, our technique tries to retrieve the related log files.

5.2.1.2 Multiple Operational Features

In the previous experiment, the log files were obtained by exercising different feature groups of the application. Hence, log files corresponding to different feature groups are likely to have few common events. Only a few events logged by entry point and exit point modules common to different feature groups will be seen in multiple log files. All other events among those logs would be different. Naturally, event logs having only a few common events represent vastly different distributions. It is possible to believe that this bias can result in seeing higher effectiveness of our technique, which is unlikely to exist in real world. Hence, we conduct this experiment, having incremental addition of feature groups to the log files.

We start this experiment with exercising a single feature group of the application using the workload generator, and collect the log files. For subsequent log files, we keep adding the feature groups one by one to the list of exercised feature groups. Thus, we build a repository of seven log files which represent operational profiles with incremental feature groups exercised in those profiles. Now we rerun those workloads three times each. Thus we have a pool of 28 log files for each which, our technique tries to retrieve the related log files. Now we apply our technique to retrieve the relevant log files. For each log file, the expected relevant log files are its three siblings from the three reruns, followed by the neighboring log files in which one less and one more feature group was exercised.

5.2.1.3 All Feature Groups

In the previous experiment, each log file had a mix of feature groups exercised in it. However, because the feature groups were exercised incrementally, it is obvious that each log file would exhibit successively more events. Thus the log files are likely to have different set of events. The set of distributions which have different set of events are likely to show greater distance, compared to the set of distributions with common events. It is arguable that this can result in seeing higher effectiveness of our technique in such situations, which are unlikely to exist in real world. Hence, we conduct this experiment with real world operational profiles.

The log files in this experiment have all the events in common, but differ only in the frequencies of those events. We conduct multiple load sessions on the application, and exercise all the feature groups. We make the load sessions to differ only in the intensities of exercising the feature groups. We conduct three load sessions with varying intensities of the seven operational features. Then we rerun each of the seven load sessions three times each, to obtain log files which represent similar operational profiles. For each of the 12 log files, the expected relevant log files are its three siblings from the three reruns.

5.2.1.4 Real World Log Files

In the previous experiment, each log file had a mix of operational features exercised in it. However, because the operational features were exercised incrementally, it is obvious that each log file would exhibit successively more events. Thus the log files are likely to have different set of events. The set of distributions which have different set of events are likely to show greater distance, compared to the set of distributions with common events. It is arguable that this can result in seeing higher effectiveness of our technique in such situations, which are unlikely to exist in real world. Hence, we conduct this experiment with real world operational profiles.

We apply our technique on execution logs from three deployments of the application. However, we do not know the expected result set, unlike the previous two experiments. So we divide each log file in four segments, for which relevant log files are being retrieved. Assuming that usage pattern for any field deployment will not change to a great extent in short duration, we expect that our technique should retrieve the three segments of the same log file for each of the 12 log file segments.

5.2.1.5 Combining all the Log Files

In this final experiment for operational profiles, we compare together all the log files generated in all the previous experiments. As a result, we have some log files that exhibit different operational feature, some exhibiting incremental addition of operational features, and some have same operational features, but different intensities. This experiment includes all possible scenarios and a large pool of log files to be compared. It represents the most intense test of accuracy of our technique. In total, we have 80 different log files profiles – collection of all the log files listed in sections 5.2.1.1 to 5.2.1.4. For each log file, the expected relevant log files are same as described in those sections.

5.2.2 Studying Retrieval by Signature Profile

Experiment Setup

The experiment setup for studying signature profile retrieval needs log files with similar signature events, as well as log files with different signature events. The *startup events* are a set of known signature events in the log files of the application under study. The startup events are logged by the application at the application startup. The startup events log the state of the environment, such as list of processes running in the system, system uptime, and configuration parameter values. To use the startup events as signature events of

the log files, we split each of the log files in four segments. Hence the first segment of each log file contains the startup events, while the subsequent three segments do not have those. For each first segment of each log file, the expected relevant log files are the first segments of other log files. For each the first segments, the remaining segments of the same log file are likely to be operationally similar, but we do not expect those in the result set as we are trying to retrieve log files based on similar signature profile.

Experiment Results

We took all the log files from the previous study on the operational profile, except the reruns. Thus we have 20 log files, which are divided in four segments each. We applied signature profile based retrieval technique on the first segment of each of the log files. Our technique correctly retrieved the first segment of other logs with 100% precision and 100% recall, even though the first segment is likely to be operationally closer to the other three segments of the same log.

6. Discussion of Results and Limitations

The experiments discussed in previous subsections demonstrate the performance of our technique. We achieved perfect results for the DS2 application because of the simplicity of the application and balanced instrumentation of all the operational features of the application. For the industrial application, our technique for operational profile performed well with the K-L divergence metric, and marginally better with the cosine distance metric. We believe the inaccuracies in the results for the industrial application stem from the complexities of real world applications, as listed below:

1. Real world applications often log a large number of events which do not correspond directly to a particular operational feature, such as idle time events, server health check events, and startup and shutdown events. Moreover, there can be an imbalance of such events, which can lead to inaccuracies in the result of our technique. For instance, if the application generates the health check events more frequently while in idle time, this is an example of imbalance.
2. Another root cause of inaccuracies in the industrial application can stem from the imbalance in the number of events per feature. As the exact event-to-feature mapping is not known, our technique cannot detect such issues. One simple way to handle such wide imbalances is to create meta-events which group co-occurring events together. These meta-events can be used for measuring the distance between event distributions.

Empirical research studies should be evaluated to determine whether they are measuring what they were designed to assess. In particular, we should examine if our finding that a given log file is more relevant to a particular log file compared to others is valid and applicable in general; or if it is due to any flaws in our experimental design. Four types of tests are used [9]: construct validity, internal validity, external validity, and reliability.

Construct Validity Construct validity is concerned with the meaningfulness of the measurements – Do the measurements quantify what we really intend to measure? We claim that locating related execution logs attached to customer engagement report will help support analysts in resolving problems sooner. We have not validated this claim, but based on our experience, locating a relevant case is usually of great value and provides many starting points if not the needed final solution.

Precision and recall metrics do not capture the internal rank among the retrieved operational profiles. For example, consider that our technique retrieved OP2, OP3 and OP4 (in that order) but the actual rank of closeness among these three is OP3, OP2 and OP4 (in that order). The precision and recall metrics do not seem to reflect such unfairness in retrieving OP3 first instead of OP2. In our experiments, we did not observe such unfairness. Furthermore, we assume that all the related engagement reports retrieved by our technique are useful to the analyst working on a new customer engagement.

Internal Validity Internal validity deals with the concern that there may be other plausible reasons to explain our results – Can we show that there is a cause and effect relation between differences in operational profiles and ranking of those by our technique? We assume here that execution logs capture the operational profile and signature profile of an application. We believe this is a valid assumption; however, the presence of wide imbalances in event logging, as discussed above, can invalidate our assumption. Moreover, our case study uses logs from the same version of an application. We did not test our technique on the execution logs of different versions. We believe limitations might be observed if there are large changes in the type of logged events.

External Validity External validity tackles the issue of the generalization of the results of our study – Can we generalize our results to other software applications? Although we applied our technique on a small test application and a complex enterprise application developed by a large number of practitioners, we only looked at two applications. Therefore our results may not generalize to other types of applications.

Reliability Reliability refers to the degree to which someone analyzing the data would reach the similar results as us. We believe that the reliability of our technique is very high. Practitioners and researchers can easily run the similar tests on their applications (or the

DS2 application) to produce findings specific to these applications, and compare those to our findings.

7. Conclusion and Future Work

Retrieval of relevant engagement reports helps support analysts resolve client issues quicker and better. Retrieval of relevant engagement reports is based on similar operational and signature profiles. We presented a technique to analyze the execution logs from the customer engagement repository and retrieve the relevant execution logs and corresponding customer engagement reports. Our technique can equally aid in remote issue resolution by identifying relevant engagement reports and recommending resolution steps.

Our technique can be applied immediately on an application, since the execution logs of most applications are readily available and are usually archived in the customer engagement repository. It requires no code changes, nor does it require any data collection from customers. Hence it can be easily adopted by companies and does not depend on a particular software application, version, build, or platform. We plan to apply our technique on other software applications to generalize our findings across different types of software applications.

Acknowledgement

We are grateful to Research In Motion (RIM) for providing us access to large enterprise applications and the deployment execution logs used in this paper. The findings and opinions expressed in this paper are those of the authors and do not necessarily represent or reflect those of RIM and/or its subsidiaries and affiliates. Our results do not in any way reflect the quality of products and services offered by RIM, its subsidiaries or affiliates.

8. References

[1] S. C. Hui and G. Jha, Data mining for customer service support, *Inf. Manage.* 38, 1 (Oct. 2000), 1-13.

[2] S. Elbaum and S. Narla, A methodology for operational profile refinement, *Proceedings of the Annual Reliability and Maintainability Symposium*, 22-25 Jan. 2001, pp. 142 – 149.

[3] J. H. Andrews, Testing Using Log File Analysis: Tools, Methods, and Issues, *Proceedings of 13th IEEE International Conference on Automated Software Engineering* (October 13-16, 1998), ASE 1998, pp. 157-166.

[4] Apache Logging Services Project. http://logging.apache.org

[5] D. A. Menascé, V. A. Almeida, R. Fonseca, and M. A. Mendes, A methodology for workload characterization of E-commerce sites, *Proceedings of the 1st ACM Conference on Electronic Commerce* (Denver, Colorado, United States, November 03 - 05, 1999), EC '99, ACM, New York, NY, pp. 119-128.

[6] A. Hamou-Lhadj and T. C. Lethbridge, A survey of trace exploration tools and techniques, *Proceedings of the 2004 Conference of the Centre For Advanced Studies on Collaborative Research* (Markham, Ontario, Canada, October 04 - 07, 2004), IBM Press, pp. 42-55.

[7] P. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining, (First Edition)*, Addison-Wesley Longman Publishing Co., Inc., 2005.

[8] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, John Wiley, New York, 1991.

[9] R. K. Yin, *Case Study Research: Design and Methods.* Sage Publications, Thousand Oaks, CA, 1994.

[10] Zimbra Collaboration Suite. http://www.zimbra.com/products

[11] Microsoft Exchange Server http://www.microsoft.com/exchange/default.mspx

[12] S. Ramanujam, H. E. Yamany and M. A. M. Capretz, An Agent Oriented Approach to Operational Profile Management, *International Journal of Intelligent Technology Volume 1 Number 4*, 2006.

[13] DIRECTIVE 2006/24/EC OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL. http://europa.eu.int/eur-lex/lex/LexUriServ/site/en/oj/2006/l_105/l_10520060413en00540063.pdf

[14] D. Jaffe and T. Muirhead, The Open Source DVD Store Application. http://www.dell.com/downloads/global/power/ps3q05-20050217-Jaffe-OE.pdf 2005.

[15] W. Weimer and G. C. Necula, Mining Temporal Specifications for Error Detection, *Proceedings of Eleventh International Conference on Tools and Algorithms for the Construction and Analysis of Systems* (April 4-8, 2005), TACAS 2005.

[16] Summary of Sarbanes-Oxley Act of 2002. http://www.soxlaw.com/

[17] Z. M. Jiang, A. E. Hassan, G. Hamann, and P. Flora, An Automated Approach for Abstracting Execution Logs to Execution Events, to appear in the *Journal of Software Maintenance and Evolution: Research and Practice*, Sept. 2008.