Analysis of Execution Log Files

Meiyappan Nagappan
North Carolina State University
890, Oval Drive,
Raleigh NC 27695-8206, USA
(001)919-513-5082
mnagapp@ncsu.edu

ABSTRACT
Log analysis can be used to find problems, define operational profiles, and even pro-actively prevent issues. The goal of my dissertation research is to investigate log management and analysis techniques suited for very large and very complex logs, such as those we might expect in a computational cloud system.

Categories and Subject Descriptors
D.2.5 [**Software Engineering**]: Testing and Debugging – *Debugging aids, Diagnostics;* D.2.10 [**Software Engineering**]: Design – *Representation;*

General Terms
Algorithms, Design, Economics, and Standardization.

Keywords
Log files, Analysis of textual data, diagnosis, fault isolation.

1. INTRODUCTION
Modern software-based systems collect information about their activity in logs. The information in the logs can therefore be used to trace the provenance of the system's activities. Logs can be single files, collections of files, databases, or streams of data. The term 'to log' is analogous to making entries in a logbook to keep track of activities completed. The information in the logs typically consists of timestamps, descriptions of events or actions, system state information and/or error information. Each log record or entry may also contain user information, and application information. Logs are most often collected for system monitoring, system debugging, identifying security and privacy violations and fault diagnostics purposes. Examples of latter are: *fault diagnosis, fault detection by monitoring* [1, 4]*, fault isolation* [6], *fault prediction* [10], *operational profiling* [5, 8] etc. Numerous tools and techniques are available to carry out log analytics. The potential applications are constantly increasing as noted by Oliner and Stearley [9].

In my exploratory research so far, intended to gain insights into the log management and analysis issues and formulate solution components and options, I have focused on log files generated by the NC State's Virtual Computing Laboratory (VCL). This is a university-wide 24x7 private cloud computing solution serving 30,000+ students. Log files from these systems are large, complex, and dynamic. The system produces hundreds of MBs of log data with millions of events in them. Full manual analysis of these logs is not a realistic option, yet they need to be used daily by the VCL developers and managers to resolve problems. Unfortunately, existing log analyses tools leave a lot to be desired. They have issues with varying formats, varying granularity of the collected information, the noise inherently present in logs files etc. Scalability of the analysis tools and techniques is of utmost importance. When analysis rate is slower than the rate at which information is collected in log files its, usefulness is reduced. I believe that current log analysis methods are too ad hoc and do not scale well enough to be effective in the domain of large logs (for example generated by cloud systems). Properly designed logging and log analyses will gain significant attention in this context as tools for analysis of system performance, usage, security etc.

ICSE '10, May 2–8, 2010, Cape Town, South Africa.

1.1 Goals
The goals of this research are

a) Framework: to develop a component-based adaptable end-to-end framework for the analysis of logs,
b) Abstraction and Analytics: to develop examples of pro-active, scalable and adaptive abstractions and analytics suitable for very large scale and very complex logs, and
c) Evaluation: to research assess, scalability, adaptability, usability, effectiveness and efficiency metrics for evaluation of the log processing and information extraction algorithms.

The framework will take into consideration that different users look for different kinds of information in the same log files. This work includes development of algorithms for each individual framework component – for example algorithms for efficient and accurate log abstraction, fault diagnosis and log summarization. Evaluation includes theoretical and simulation-based evaluation of the scalability of these algorithms, experimental verification of the theoretical results, including calculation of precision and recall, evaluation of the usability of the system.

In this paper I also discuss related work, and some of my early results. The latter includes a novel linearly scaling algorithm, based on the suffix array data structure that I developed to determine operational profile of a system using its execution log files. I also present a plan for future research that includes completion of the framework, associated analytics and its effectiveness and efficiency evaluation criteria. I intend to complete my PhD dissertation research by December 2011.

2. END TO END FRAMEWORK

A significant amount of activity needs to precede successful data mining of log files. This includes log data collection, log abstraction (or reduction of information to more canonical or generic structures and concepts), and formatting of the data for use by analytics engines (see Figure 1). Currently, most of these activities are specific to a particular system or application, and often they are conducted in a very ad hoc manner. I propose to componentize the principal log management and analysis task groups and build a log analysis framework into which individual tools and techniques from each of these phases can be plugged into and allowed to interact through common application interfaces.

Of course, the concept of abstraction or analysis of log files is not a new one. *Sophisticated analytics – simple abstraction*: In the literature there are numerous log analysis techniques, each one with their own abstraction approach [e.g., 8,10,12]. While the analyses techniques tend to be highly sophisticated, the abstraction processes are usually very simple. Unfortunately, often simple abstraction techniques may not work beyond the specific application they have been designed for. *Sophisticated abstraction – simplistic analytics*: On the other end of the spectrum are tools like Spunk [1], and Swatch [4] that have highly sophisticated algorithms to extract and abstract the information from the log files, but the analytics they offer are simplistic. In addition to developing new and more efficient algorithms for each component, we intend to evaluate and document the best combination of new and existing tools in our framework for each of the analysis goals.

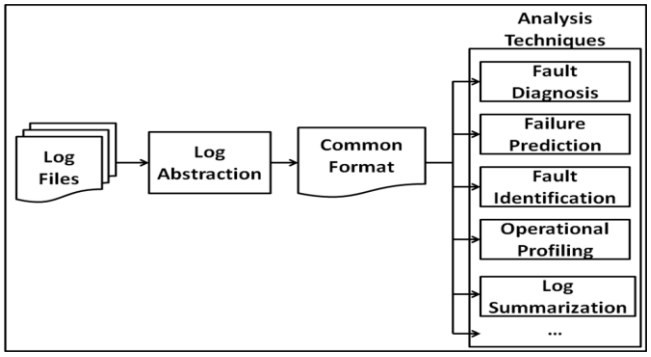

Figure 1. Framework for Analyzing Log Files

2.1 Log Abstraction

The first component in our framework is the log abstraction component. This is where we have tools and techniques to abstract the given log file to a common format. The motivation for log abstraction is that often the information in a log file contains a lot of information that comes from dynamic run time changes. Some examples are IP-addresses, port numbers, process identifiers etc. Also in this category are timestamp fields, static message type fields, and a variable parameter fields. While that information, of course, is useful it also makes it very diverse. It is possible to make the log structure more canonical by abstracting some of the specificity and variability. The presence of the variable parameter fields makes it difficult to reason about the data in an automatic fashion. So construction of a higher level conceptual data element may be helpful. An example in the network traffic space is 'flows'. A TCP flow log consists of source and destination IP numbers, port numbers, time stamps, direction, payload size, etc. Talking about such flows may be more useful for identification of security issues than talking about individual IP numbers or port numbers.

Even the training of data mining algorithms can be affected adversely when too much detail is thrown at them. Therefore, tools and techniques may be required to smooth the "noise" and abstract each line in a log file to a unique message type. One can recognize two types of "noise". The first one is variable parameter information mentioned above. The second type is a sequence of events that always happens together (e.g., a "flow" always has source followed by destination). For analyses such event-based clusters can be more useful and more efficient, than individual events, as the log files can often have a great deal of information.

Examples of the log abstraction techniques found in the literature include the Simple Log File Clustering Tool [11], the Levenshtein's edit distance technique [10], and Xu's approach of leveraging the source code [12]. These techniques are used for abstracting the lines in the log file to an integer or a unique message type. Clustering algorithms are used to cluster very frequently occurring set of events. These algorithms are used to smooth the data, i.e., reduce the "noise" in the log files. This component of the framework translates the information in the log file to a common form so that various analysis algorithms can operate on it. I intend to look at graph mining algorithms for this component.

2.2 Common Format

The next element in the framework is the log file in a common format. The fields of interest include an explicit timestamp, the static message type, dynamic variable parameter information, and an integer that uniquely identifies the message type for each line in the log file. This last field is required as most data mining and analyses techniques work better with integers rather than text. There could be a separate file with the (message type, unique identifier or ID) tuple. A comma separated value format is suggested. The ID could also contain information about the location in the source code from which log information originally came from. The information used by most analysis techniques is a subset of this information. Each operates on a slightly differing format. New tools and techniques could either directly use the data in this format or write a small parser to convert the data from this format to the format in which they would want the information.

2.3 Analysis Techniques

Once abstraction pre-processing to a common format is complete, a variety of analysis techniques can be applied to provide the user with information the user seeks. These analysis algorithms are the next set of components in my framework. The analyses techniques are evaluated on, for example, the precision, recall and utility of the results as well as the efficiency in calculating these results.

In their research Hamou-Lhadj and Lethbridge [3], present a variety of trace exploration tools and techniques that are used to analyze the information in the log files. Different users have

different needs, so I first define the classes of users. They are i) high level users of a system who generally do not look into a log file, ii) system administrators who look into the log file to find a workaround for the failure that the high level users have faced, iii) the technical assistance people or developers who want to find the root cause of a problem to either provide a solution for the system administrators or fix the bug in the source code. Table 1 gives examples of the analysis technique goals, the kinds of users that would need it and the status of my thesis research in this context. In subsections that follow I discuss the different kind of analyses goals that different classes of users would need.

Table 1. Analytics

Analysis Technique Example	Users	Status
Operational Profiling	Program Managers	In progress [8]
Failure Prediction	System Administrators	In Progress
Log Summarization	High Level Users	To be Done
Fault Diagnosis	Technical Assistance and Developers	To be Done

2.3.1 Operational Profiling

Operational profiling [7] is an example of the analysis that a user may want to do. Techniques based on information from log files, like the one proposed by Hassan et. al. [5] and Nagappan et. al. [8], can provide a greater insight for the software engineers. In [8] I proposed a novel algorithm to extract the operational profile of the system from its execution logs. The approach is completely automatic, scales linearly, and collects the full range operational profile from the most frequently used parts of the system to the least frequently used parts of the system.

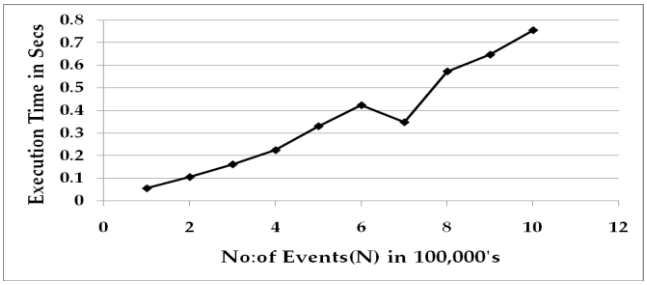

Figure 2. Results for the Operational Profiling Algorithm

In [8], we use the Suffix Array (SA) and Longest Common Prefix (LCP) array of the abstracted log file. Using these data structures we are able to very efficiently calculate the frequency of every sequence of event patterns. Theoretical analysis of the space and time complexity of our algorithm shows an almost linear scaling. We experimentally evaluated this claim using VCL logs. The data is plotted in the graph shown in Figure 2. We conducted an interview with the VCL Program Manager to evaluate the accuracy and utility of the technique. The sequence of events that our algorithm produced as the most frequent was expected, but to the surprise of the manager some other tasks featured in the top ten frequent sequences were not. They were able to optimize the system based on our results. Our algorithm also picked out the least frequently occurring events. The program manager used these results to test system scheduling algorithms further.

2.3.2 Failure Prediction and Identification

Failure prediction may be another analysis desired by a user. In failure prediction we try to use the current set of events to predict a failure that may occur later and take some preventive measures. The system administrators might use online failure prediction tools like the one proposed by Salfner and Tschirpke [10], but they might also want to identify anomalous behavior of a system using the log files. For this they might use failure identification tools like Splunk [1] or Swatch [4]. We are currently working on a failure prediction method for dynamically changing the amount of logs collected.

2.3.3 Log Summarization

Summarization is yet another operation that may be needed. High level users may not understand what they see in a raw log file. For these kind of users a summary of what happened in the system beyond the obvious results of a failure is useful. For example, when system crashes it may request to send an error report. But often the reason is not given. It may be a commonly occurring problem, in which case the analyses technique could operate on the log file and give a simple English description of the problem based on historical data. I intend to work such a solution as part of this research. The initial ideas include a text summarizing tool and a categorizing algorithm.

2.3.4 Fault Diagnosis

Fault diagnostics are probably the most common use of logs. In fault diagnosis we know that a failure has occurred and we try to find the root cause and the fault (or physical defect) that caused the failure[1]. The developers and technical assistance people typically use a database of sequences of events for known problems. They search the database for the sequence of events that a customer submits to them. These users want to identify the fault in the source code that caused the failure so that they can fix the source code or just provide a workaround, and they will need fault diagnosis techniques like the one by Mariani and Pastore [6]. I intend to identify and include into the framework a suite of efficient fault diagnosis algorithms.

3. EVALUATION

As part of this work, I plan to identify metrics and evaluation criteria suitable for assessment of the efficiency and effectiveness of the framework components. Specifically i) For each algorithm I will theoretically estimate upper bounds for space and time complexity. This will dictate the choice of algorithms as scaling, for example, is an important issue in log file analyses. ii) I will experimentally test the theoretical findings on log files from real

[1] A user mistake results in an **error** that causes a physical anomaly, a **fault**, in a software artifact. This initial fault can propagate and generate a series of **defects** in subsequent software stages. When run-time execution with specific inputs and in a specific environment encounters such a defect, it may put the system into an (internal) **error state**. If this error-state results in an observable **anomaly** that does not meet requirements/specifications and/or end-user expectations, and it is confirmed as such, we say we have observed a **failure**. For example, security failures can be traced to many causes, from physical defects, to lacks in specifications, to abnormal or unexpected **operational profiles**.

world applications. iii) I will then evaluate each algorithm on the accuracy of the results (by calculating the precision, recall, etc.).

I plan to evaluate a number of relevant techniques found in the literature before including them in the framework. The goal is to find the best configuration of tools and techniques based on the goal of the analyses. One of the elements will be the ease of use of the tools and techniques in a particular configuration of the framework (e.g., by conducting interviews with the users of a particular analysis technique). The result would be a collection of configurations for the framework based on the users of the analysis technique.

4. TIMELINE

Until now I have spent most of my effort understanding the log abstraction and analysis problem. Based on this exploratory work, I have developed the framework concept (see Figure 1), and I have started in-depth investigation of the issues related to individual framework components. An example is the operational profiling algorithm I describe in [8]. The log abstraction algorithm is similar to the log abstraction algorithm in Xu et. al's research [12]. I used the source code to identify the static message type and dynamic parameter fields. Then I assign a unique integer ID to each message type. Thus I abstract the log file to a set of integers.

Recently I started looking into the use of a logistic regression model for predicting failures by examining a window of events in the log file. The intention is to dynamically change the level of logged information. When a fault is predicted, more information is collected to enable a better fault diagnosis. I expect to prepare a paper on this topic in the January 2010 timeframe.

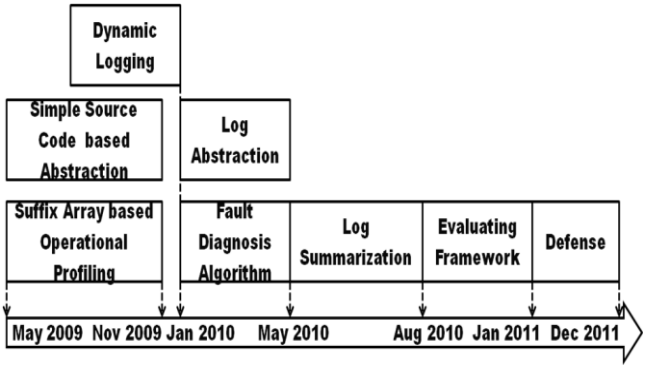

Figure 3. Timeline of Research

So far the log files that I have been using to experimentally explore and verify solutions are from VCL [2], but I am in discussion with two very large industrial organizations about use of their data to assess the framework and fault diagnosis algorithm that would help real customer problems with greater efficiency. The algorithms should enable the users perform a root cause analysis based on the log files from the customer. I intend to complete this by May 2010. This work would be followed by log summarization investigation. The goal is to provide an appropriate summary of circumstances surrounding special events of interest, such as a failure, and perhaps offer to end-user advice about the cause and a workaround in the case of commonly occurring problems. The end goal is to reduce the number of customer calls to technical support centers until a software fix can be distributed. I intend to complete this research over the next summer and submit a paper by August 2010.

While identification of appropriate metrics and evaluation of the framework is an ongoing process, I plan to focus on it as the final step in the proposed work. The idea is to identify which combination of tools works best for a particular analysis objective. This would identify near optimal configuration for each kind of user. I intend to complete this by January 2011.

5. ACKNOWLEDGMENTS

I would like to thank Dr.Vouk for advising me on this dissertation topic. I would also like to thank Dr.Stallmann, Dr.Samatova, Dr.Heber, and Ms.Boyer for their insightful discussions on data mining related topics. This research was funded in part by the DOE grants DE-FC02-ER25809, and DE-AC02-05CH11231.

6. REFERENCES

[1] Splunk http://www.splunk.com/ (accessed 11/20/2009).

[2] Virtual Computing Lab http://vcl.ncsu.edu/ (accessed 11/20/2009).

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

[4] S.E. Hansen, and E.T. Atkins. 1993. Automated System Monitoring and Notification with Swatch. 7th USENIX Conference on System Administration. Berkeley, CA. November, 1993. 145-152.

[5] A. E. Hassan, D. J. Martin, P. Flora, P. Mansfield and D. Dietz. 2008. An Industrial Case Study of Customizing Operational Profiles Using Log Compression. In ICSE `08 Leipzig, Germany, May 10-18, 2008, 713-723.

[6] Mariani, L.; Pastore, F. 2008. Automated Identification of Failure Causes in System Logs. 19th International Symposium on Software Reliability Engineering, 10-14 Nov. 2008. 117-126.

[7] J. D. Musa., 1993. Operational profiles in software-reliability engineering. IEEE Software, 10(2):14–32, 1993.

[8] M. Nagappan, K. Wu, M.A. Vouk,. 2009. Efficiently Extracting Operational Profiles from Execution Logs using Suffix Arrays. 20th International Symposium on Software Reliability Engineering, 16-19 Nov, 2009, India. 41-50.

[9] A. Oliner and J. Stearley., 2007. What Supercomputers Say: A Study of Five System Logs. 37th Annual IEEE/IFIP international Conference on Dependable Systems and Networks. Washington, DC. June, 2007. 575-584.

[10] F. Salfner and S.Tschirpke. 2008. Error Log Processing for Accurate Failure Prediction. In Proceedings of the First USENIX Workshop on the Analysis of System Logs, December 7, 2008, San Diego, CA, USA.

[11] R. Vaarandi. 2003. A data clustering algorithm for mining patterns from event logs. IEEE Workshop on IP Operations and Management, 1-3 Oct. 2003, 119-126.

[12] W. Xu, L. Huang, A. Fox, D. Patterson, and M. Jordan. 2008. Mining Console Logs for Large-Scale System Problem Detection. SysML`08, Dec 2008. 1-6.