Using Component Interaction Model and Network Traces for Root-cause Analysis*

Atul Kumar
IBM Research
Manyata Embassy Business Park, Nagwara,
Outer Ring Road, Bangalore, India
kumar.atul@in.ibm.com

Anil Nair
Toshiba Software India Pvt Ltd
Fortune Summit, 6th Sector, HSR Layout,
Hosur Main Road, Bangalore, India
anil.nair@toshiba-tsip.com

ABSTRACT

Root-cause analysis after a system failure/error is an important activity to determine exact reasons for failure/error. Most of the time, these error conditions cannot be reproduced or it is not feasible to run the system again using the exact same scenario. Therefore, execution trace log of various functions/components recorded during the event is essential for root cause analysis and debugging in a complex system. Source code level instrumentation for dynamic analysis provides accurate execution trace log. But it is difficult to use an instrumented system in production environments because of performance and system stability issues. In a distributed system, intercepted network messages can be analyzed to identify interactions between various components of the system. However, messages captured on network alone do not provide complete information because messages between components on same host would not appear on network. We present a new idea to construct interaction information among components of a distributed application using messages captured on network and an interaction model that is a set of rules and heuristics about component interaction. An interaction model is pre-built offline using profile information and static control flow graph of the system. Profiling is done with test data in a non production environment such as a test environment using 'close-to-real' test scenario. Messages corresponding to components interaction are captured on network to create a partial execution trace log. Then the trace log is completed using the pre-built interaction model.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.2.c [**Software Engineering**]: Distributed/Internet based software engineering tools and techniques

General Terms

Component Interactions, model based analysis

*This work was done when authors were with ABB Corporate Research

Keywords

Component Interactions Model, Dynamic Analysis, Root-cause analysis, Network Packet Filtering and Analysis

1. INTRODUCTION

Execution trace log data[1] is useful for debugging and root cause analysis by identifying sequence of operations that led to a system failure or error. Component interaction and information flow is also used to identify performance bottlenecks in a complex system. These logs are also useful in finding better deployment plan for distributed components on different hosts by looking at inter-host component interaction patterns. Additionally, it can also help in optimizing and prioritizing test cases.

Such logs can be obtained using dynamic analysis [7]. Original source code is instrumented by inserting instrumentation code to log some information at desired points (normally at the beginning and the end of a function). Then application is executed and log generated by the instrumented code is logged. However, it may not be possible to use instrumented applications in a production environment. Instrumentation code causes overhead that may not be acceptable in a production environment because of performance reasons. Controller hosts in automation systems have limited logging and tracing capabilities because embedded devices are normally resource constrained (memory and/or CPU time) and may not bear this overhead. Using instrumented code as created by coverage tools or debug tools slows execution of programs to a degree that these tools are not usable in production environments or even in complex system test environments. Moreover, it may not be reasonably safe to execute instrumented code in live production environments because logging of system activities may cause exceptions that may destabilize an otherwise stable system.

In a large system, components of application are often distributed on several hosts in a system. Interactions between components of such applications take place via network communication. We propose an idea for a system where messages on network are eavesdropped. Since all interactions among components of an application do not appear on network (e.g., communication between components on the same

[1]Temporal information about the start of execution and the end of execution for functions/components in a program/system and also what function/component execution followed/precedes what other function/component execution.

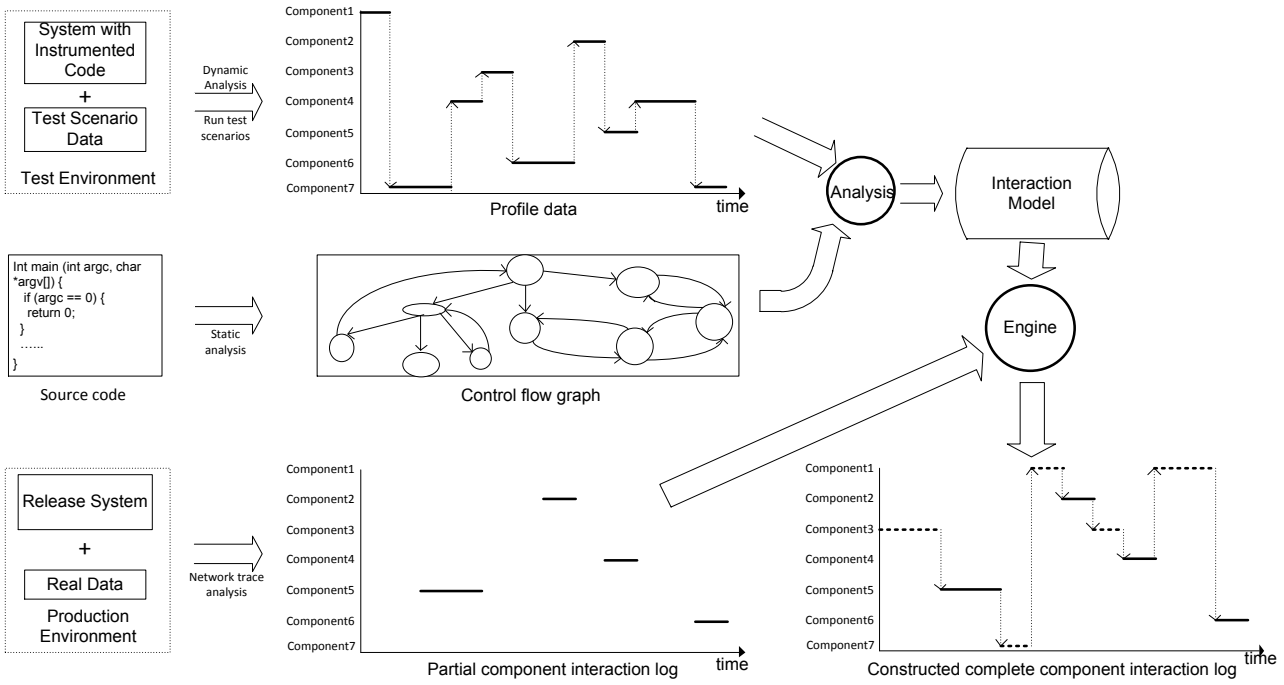

Figure 1: Process of Constructing Execution Trace Log

host), component interaction information captured on network therefore will be incomplete. To construct a full execution trace log from partial component interaction information built using network messages, an interaction model is used. The interaction model is built beforehand using control flow graph of the application obtained from static analysis and profile information collected in a test environment using instrumented application and test scenario.

2. CONSTRUCTING EXECUTION LOG FROM NETWORK MESSAGES

2.1 Capturing Network Messages

Several tools exist to capture and analyze messages on data networks. Normally, these tools require a host running them to be present on the same network on which messages need to be listened to. Intercepting network messages between different hosts in a distributed system using a separate computer has no impact on system performance. Tools such as Netmon [1], Microsoft Message Analyzer [2], Tcpdump/libpcap [4], WinDump/Winpcap [5], Wireshark [6], etc. make it easy to analyze network packets at various network layers including application layers. Packets can be filtered for specific patterns. Applications developed using popular frameworks such as Dot Net, J2EE, CORBA etc., have well defined message formats for sending messages between components and are easy to identify automatically using packet filters.

2.2 Creating Execution log

Following are major steps in the process of constructing execution log from network messages.

1. Instrument source code using traditional methods, use comprehensive and near real test scenario to gather profile data. Use dynamic analysis to construct trace logs for all test scenarios.

2. Perform static analysis on source code to generate control flow graphs of the system.

3. Use the above two to create an interaction model. This model is used to identify patterns and create heuristics about various work-flows in the system. If some information is missing in a execution trace log, then the model should be able to tell most likely candidates for missing places. When used with temporal information, this model should be enough to construct a complete trace-log form the partial log.

4. In a production environment, capture network messages to identify component interactions. Create a partial execution trace log using this information.

5. Use the interaction model created in step 3 to generate complete execution trace log.

Figure 1 shows the process to construct execution log using network traces and a previously built patterns and heuristics.

The top left part of diagram shows step 1 of the process mentioned above. Output of dynamic analysis for each test

scenario is used later to build an interaction mode. The middle left part of diagram shows static analysis process that generates control flow diagram from source code. The top right part of diagram shows the process of creating interaction model. This model essentially holds a set of rules and heuristics for various possible interactions among components of the system. The bottom left part of diagram shows the process of creating partial component execution trace log from network messages. This log is used by an engine shown in the bottom right part of diagram to construct complete execution trace log using rules and heuristics from the interaction model.

2.3 Interaction Model

Building a good interaction model is key to success of this idea. Control flow graphs for various modules/components of an application provide all possible interactions in applications. Interaction between modules/components can be obtained from call graph, input data, and system integration model. This is still not sufficient to capture dynamic behavior of application. For example, if there are n possible paths an execution sequence can take from a particular point, then temporal information can reduce that possibility to k (where $k << n$). System may follow a different execution sequence at start-up, at user input time and at time of I/O. With a good set of test data, a dynamic interaction model can be built which can cover most common usage scenario. Combining static and dynamic interaction models can reduce total number of possible execution paths. Some heuristics built around execution behavior and input values are used to select the most likely path.

2.4 Assumptions

This approach makes the following assumptions.

- Network messages are not encrypted.
- Systems under considerations are distributed systems where significant interaction among components passes over network.
- Sufficient test scenario data is available that is close to real usage scenario.
- Interaction model is rebuilt after there is any change in the system.

3. RELATED WORK

Performing root-cause analysis in distributed system is a well studied subject. A recent work on run-time root cause analysis in distributed systems is presented in [10]. This work addresses problem of deriving relationships for fault correlation in adaptive distributed systems where components are dynamically installs/updates/removes and presents a state chart-based solution which tried to identify the sequence of method execution.

An approach to combine model-driven techniques with runtime models to perform root cause analysis of executing systems is presented in [11]. The approach is to combine advantages of model-driven development with reusable software artifacts. Interactive visualizations enable efficient tracing of log file entries and corresponding model artifacts during runtime.

Some tips and tricks to help troubleshooters extract root cause information from network traces is provided in [3]. Objective is to reduce what might be hundreds of gigabytes of data to essential events that show root cause of a problem. Focus is to use network traces and then narrow the fault down to a box.

An execution environment for Java programs is presented in [8] that improves execution performance by using both online and off-line profile information to guide dynamic optimization. A dynamic compilation system based on JikesRVM was developed that makes use of both.

A model-based diagnosis approach is discussed in [9] that discovers faults based on generic fault models and abstract event traces. These events may be associated to multiple system components. Availability of fault for each component is not assumed and generic fault models of classes of faults are used instead.

Our proposed idea is different from above works because, in our approach, we construct a trace log very close to the one obtained from dynamic analysis of instrumented application without actually instrumented it. We rely on maturity of a pre-built model but it needs to be validated by actually building a prototype tool based on our idea and then comparing the trace-log generated by our tool with complete data captured from instrumented application.

4. CONCLUSIONS AND FUTURE WORK

We presented an idea to construct components interaction trace log for components of a distributed application in live production environments. An interaction model is first built offline by generating profile data in a test environment. Then, in a live production system, a partial components interaction trace log is created from network messages eavesdropped from a separate host on the same network. Finally, a complete execution trace log is constructed by an engine using partial logs and interaction model.

We plan to start a short project on this idea. Purpose is to validate our hypothesis presented in this paper that the execution trace log can be constructed by only capturing network messages in a live production system (other information required is collected offline). In particular, we would focus to find answers of the following questions.

- Does enough component interactions take place over network (between hosts) in a real distributed application? If yes, then how much is 'enough'? Can we use less messages than what can be captured on network to reduce size of network log?
- Can we develop heuristics that help in recreating complete profile from network messages and models that were built offline?
- Does log data provide enough information to isolate heisenbugs in software that are otherwise non-repeatable?

5. REFERENCES

[1] How to use network monitor to capture network traffic. http://support.microsoft.com/kb/812953.

[2] Microsoft message analyzer operating guide. http://technet.microsoft.com/en-us/library/jj649776.aspx.

[3] Network trace analysis strategies. http://www.advance7.com/wp-content/uploads/2012/11/Network-Trace-Analysis-Strategies-Whitepaper.pdf.

[4] Tcpdump/libpcap. http://www.tcpdump.org/.

[5] Winpcap. http://www.winpcap.org/.

[6] Wireshark. http://www.wireshark.org/.

[7] T. Bell. The concept of dynamic analysis. In *ACM SIGSOFT international symposium on Foundations of software engineering*, pages 216–234. ACM SIGSOFT Software Engineering Notes, November 1999.

[8] C. Krintz. Coupling on-line and off-line profile information to improve program performance. In *international symposium on Code generation and optimization*, pages 69–78. IEEE Computer Society, March 2003.

[9] W. Mayer, X. Pucel, and M. Stumptner. Diagnosing component interaction errors from abstract event traces. In *23rd Australasian Joint Conference on Advances in Artificial Intelligence*, pages 496–505. Lecture Notes in Computer Science Volume 6464, December 2010.

[10] A. Raj, S. Barrett, and S. Clarke. Run-time root cause analysis in adaptive distributed systems. In *On the Move to Meaningful Internet Systems: OTM 2013 Workshops*, pages 292–301. Lecture Notes in Computer Science Volume 8186, September 2013.

[11] M. Szvetits and U. Zdun. Enhancing root cause analysis with runtime models and interactive visualizations. In *8th International Workshop on Models at run.time*, September 2013.