Chapter 21
Non-Intrusive Adaptation of System Execution Traces for Performance Analysis of Software Systems

Manjula Peiris
Indiana University Purdue University Indianapolis (IUPUI), USA

James H. Hill
Indiana University Purdue University Indianapolis (IUPUI), USA

ABSTRACT

This chapter discusses how to adapt system execution traces to support analysis of software system performance properties, such as end-to-end response time, throughput, and service time. This is important because system execution traces contain complete snapshots of a systems execution—making them useful artifacts for analyzing software system performance properties. Unfortunately, if system execution traces do not contain the required properties, then analysis of performance properties is hard. In this chapter, the authors discuss: (1) what properties are required to analysis performance properties in a system execution trace; (2) different approaches for injecting the required properties into a system execution trace to support performance analysis; and (3) show, by example, the solution for one approach that does not require modifying the original source code of the system that produced the system execution.

1. INTRODUCTION

Challenges of using system execution traces for performance analysis. Software performance analysis is the process of analyzing performance properties (*e.g.* response time, service time, throughput) of a software system. Analyzing system execution traces is one technique used in software performance analysis. System execution traces can be generated by (1) compiling the source code of the system with instrumentation code(Wolf & Mohr, 2003); (2) collecting the log messages while executing the instrumented system (Hill J., 2010); and (3) registering for certain events in the target system and generating messages whenever that event occurs (Mod & Murphy, 2001). The first

DOI: 10.4018/978-1-4666-6026-7.ch021

method is an intrusive method because it modifies the actual source code of the target system. Second and third methods are non-intrusive, because it does not require modifying the system's original source code.

Most of the existing approaches for using system execution traces to analyze software performance are based on intrusive methods(Wolf & Mohr, 2003). The main limitation with these approaches is it requires access to the system's source code. Other approaches for using system execution traces to analyze software performance are tightly coupled with system architecture and deployment (Mod & Murphy, 2001). Finally, approaches that are not architecture-dependent require system execution traces to be generated in a certain format (Salonen & Piilil, 2012)(Salonen & Piilil, 2012),(Nagaraj, Killian, & Neville, 2012). Moreover, such approaches are not trying to utilize system log messages, but rather enforce system developers to use provided logging mechanisms. This approach therefore requires system developers to change the underlying implementation for the purpose of performance analysis. The limitations discussed above make it *hard* to generalize existing approaches for different kinds of systems, and their generated system execution trace.

We have focused on using non-intrusive approaches, such as execution log messages for performance analysis, to overcome the current limitations of using intrusive system execution traces for software performance analysis. Rather than modifying the system's original source code, we focus on creating an intermediate model to abstract out the events in the system execution trace and the relations among log messages. Likewise, we assume generated log messages are not in a certain format.

The realization of our approach is in a tool called *Understanding Non-functional Intensions via Testing and Experimentation (UNITE)* (Hill & Schmidt, 2009). UNITE uses dataflow models to describe causality relationships between event types—not event instances—in the system. This allows UNITE to operate at a higher level of abstraction that remains constant regardless of how the underlying software system is designed, implemented, and deployed (i.e., the mapping of software components to hardware components). The dataflow model is then used to process the system execution trace, and analyze performance properties.

Although it is possible to analyze performance properties via system execution traces using tools like UNITE, it is assumed that system execution traces contain several properties, e.g., identifiable keywords, unique message instances, enough variations among the same event types to support performance analysis. Moreover, the dataflow model must contain several properties, e.g., identifiable log formats and unique relations between different log formats. If planned early enough in the software lifecycle, it is possible to ensure these properties exist in both the dataflow model and generated system execution trace. Unfortunately, it is not possible to always ensure that these requirements are met.

This chapter therefore illustrates the following on adapting system execution traces and their dataflow models to contain properties required to analyze software system performance properties:

- How to adapt system execution traces and corresponding dataflow models to contain the properties required for supporting performance analysis using a framework we have developed called *System Execution Trace Adaptation Framework (SETAF)*;
- What are the different design alternatives—including their advantages and disadvantages—for adapting system execution traces to support performance analysis;
- How SETAF can be applied to system execution traces generated by different software systems; and
- A performance comparison of two different system execution trace adaptation techniques.