Industrial Case Study on Supporting the Comprehension of System Behaviour Under Load

Mark D. Syer, Bram Adams and Ahmed E. Hassan
Software Analysis and Intelligence Lab (SAIL)
School of Computing, Queen's University, Canada
{*mdsyer, bram, ahmed*}*@cs.queensu.ca*

***Abstract*—Large-scale software systems achieve concurrency on enormous scales using a number of different design patterns. Many of these design patterns are based on pools of pre-existing and reusable threads that facilitate incoming service requests. Thread pools limit thread lifecycle overhead (thread creation and destruction) and resource thrashing (thread proliferation). Despite their potential for scalability, thread pools are hard to configure and test because of concurrency risks like synchronization errors and dead lock, and thread pool-specific risks like resource thrashing and thread leakage. Addressing these challenges requires a thorough understanding of the behaviour of the threads in the thread pool. We argue for a methodology to automatically identify and rank deviations in the behaviour of threads based on resource usage.**

***Keywords*-thread-pools; behaviour-based clustering; understanding ULS systems;**

I. Introduction

Program understanding is one of the most important aspects in maintaining and evolving software systems [26]. Performance analysts need to understand the system to optimize it, quality assurance analysts need to understand the system to fix it and developers need to understand the system to upgrade it. The current state-of-the-practice of program understanding is on static understanding, based on source code and documentation, and dynamic understanding using simple use cases [1].

However, understanding how Ultra-Large-Scale (ULS) systems behave under a load is difficult because current methods typically require significant manual review of the performance data and execution logs and a high degree of program comprehension of the system [2]. This is especially problematic in high demand systems such as e-commerce and telecommunications systems, which may potentially support hundreds or thousands of concurrent connections and operations. Case studies of these types of systems have shown that failures are more often associated with an inability to scale to meet demands, leading to performance degradation, than with feature bugs [7].

Many techniques exist to produce the data necessary to understand a ULS system under a load: instrumentation and reuse of system data, such as execution logs and metrics [2]–[5]. However, instrumentation has a high overhead, so program comprehension based on logs or metrics is the most practical [6]. Logging during execution may be sparse so metrics like CPU and memory usage are often the only feasible data source, however metrics are very hard to interpret [6]. Hence, performance analysts perform an increasingly important function within the software development lifecycle.

ULS systems are designed using a variety of architectural styles and design patterns. One popular architecture is the thread pool pattern that uses pre-existing and reusable threads to service incoming requests. This architectural pattern allows the system to scale very well by facilitating both concurrent and distributed processing. This poster-paper argues for a scalable methodology to help performance analysts understand the behaviour of ULS systems that implement thread pools.

II. Poster Outline

Our poster will present:

- An overview of the challenges in understanding ULS software systems.
- A discussion of thread pools as a common design pattern in large software systems.
- Our methodology to support the comprehension of system behaviour under load.
- A case study on a large scale industrial software system showing how our methodology can
  - detect key phases in the lifetime of an executing system
  - detect and flag deviations from the common behaviour
- An overview of our on-going work in the area.

References

[1] B. Cornelissen, A. Zaidman, A. van Deursen, L. Moonen, and R. Koschke, "A systematic survey of program comprehension through dynamic analysis," *Trans. Software Engineering*, vol. 35, no. 5, pp. 684–702, Sep. 2009.

[2] Z. M. Jiang, A. Hassan, G. Hamann, and P. Flora, "Automated performance analysis of load tests," in *Proc. 25th Int. Conf. Software Maintenance (ICSM)*, Sep. 2009, pp. 125–134.

[3] M. Ernst, A. Czeisler, W. Griswold, and D. Notkin, "Quickly detecting relevant program invariants," in *Proc. 22nd Int. Conf. Software Engineering (ICSE)*, Jun. 2000, pp. 449–458.

[4] A. Hamou-Lhadj and T. Lethbridge, "Summarizing the content of large traces to facilitate the understanding of the behaviour of a software system," in *Proc. 14th Int. Conf. Program Comprehension (ICPC)*, Jun. 2006, pp. 181–190.

[5] H. Pirzadeh, A. Agarwal, and A. Hamou-Lhadj, "An approach for detecting execution phases of a system for the purpose of program comprehension," in *Proc. 8th Int. Conf. Software Engineering Research, Management and Applications (SERA)*, May 2010, pp. 207–214.

[6] H. Malik, Z. M. Jiang, B. Adams, A. E. Hassan, P. Flora, and G. Hamann, "Automatic comparison of load tests to support the performance analysis of large enterprise systems," in *Proc. 14th European Conf. Software Maintenance and Reengineering (CSMR)*, Mar. 2010, pp. 222–231.

[7] E. Weyuker and F. Vokolos, "Experience with performance testing of software systems: issues, an approach, and case study," *Trans. Software Engineering*, vol. 26, no. 12, pp. 1147–1156, Dec. 2000.