Fault Localization in Concurrent System

IN4MATX 215

Milestone 1 – Literature Review

Team Member:

Chiyu Cheng

Yi Zhou

Introduction

Testing is one of most critical and expensive tasks in the development of software. The process of detecting and locating bugs in software system is called fault localization. Manually fault localization is acceptable for the small software system but unbearable for the large-scale system. Therefore, an automated fault localization technique is necessary for the real-world complex software system. A concurrent system is one where a computation or operation can be executed without waiting for all other operations to complete. It generally involves the coordination of multi-thread/process/cores to run a program. The concurrency fault is difficult to find because a concurrent system may have large number of threads interleaving, which makes it is hard to reproduce a fault in one specific thread interleaving. The research question we have is, is it possible to locate the concurrent bugs in the large-scale system efficiently? In this project, we want to develop an evaluation framework to study and evaluate the current state of art in finding the concurrency fault. The plan is surveying 8-10 papers in this topic and evaluating the methods provided in those papers under the same standard. The basic criteria are feasibility, efficiency, and accuracy. We will test if each method is feasible and record the performance of each method on the real system like the SPLASH-2 programs from Stanford university with concurrency bug.

Based on the paper we read, the current state of arts in the field of concurrent fault localization are focused on the memory-access pattern, variable access pattern, and random execution between interleaving. The brief summaries of each paper present in the next section.

Paper Summaries

In Dr.Park’s paper[1], he provides an dynamic fault-localization technique that extends the functionality of traditional method from sequential and deterministic program to the multi-thread program. This basic idea of this technique is detecting the data access patterns in multi-thread concurrent programs. It monitors the memory access sequence in different thread interleaving to find the data access patterns and mark it with fail or success that associates with the result of program tests. Based on the data access patterns collected, the technique generates a suspicious score for each pattern to help programmer to find concurrent bugs.

In the paper “Cooperative Crug Isolation”[2], the authors provides a scalable technique to locate the root of the concurrent bugs with low-head. They use a novel approach named cross-thread random sampling strategy to reduce the overhead of this method, and track successive accesses to a memory location to keep recording the occurrence of specific interleavings at run-time. With the collection of those shared variable access, programmers can design the predicates to check whether a previous variable access is thread-local or thread-remote. Based on the fail/success result of those predicates, CCI method generates the suspiciousness score of each predicate.

In Stoller’s paper[3], he presents a novel random scheduling function to test it on a Java program. The scheduling function use heuristic to weight the random generated choice and find the choice that leads to all the reachable deadlock and assertion violations with Non-Zero probability.

In Musuvathi’s paper[4], they proposes a new algorithm that systematically search and iterate the executions of multi-threaded program with different priority order based on the number of context switch. They detects and distince the difference between preempting and non-preempting context switches of each execution, and find that the limitation of the number of preempting context-switch mitigates the explosive possible states of multi-thread executions.

In Ken’s paper[5], he proposes a randomized dynamic analysis technique with a focus on concurrent faults caused by data race condition. This technique gets the possible data race condition from an analysis tool, and use it in a random scheduler of threads to create real race conditions and resolve it in the runtime. With this technique, programmers are be able to find the real race condition on their program with low over-head.

Reference

[1] Park, S., Vuduc, R., and Harrold, M. J., “Falcon: Fault localization in concurrent programs,” in Proc. of Intl Conf. on Softw. Eng, pp. 245–254, May 2010.

[2] A Thakur, R. Sen, B. Liblit, and S. Lu. Cooperative crug isolation. In WODA, pages 35–41, 2009.

[3] S. D. Stoller. Testing concurrent Java programs using randomized scheduling. In Proc. Second Workshop on Runtime Verification (RV), volume 70(4) of Electronic Notes in Theoretical Computer Science. Elsevier, July 2002

[4] M. Musuvathi and S. Qadeer. Iterative context bounding for systematic testing of multithreaded programs. PLDI., 42(6):446–455, 2007.

[5] K. Sen. Race directed random testing of concurrent programs. In PLDI, 2008.