Title: A randomized scheduler with Probabilistic guarantees of finding bugs

Authors: Sebastian Burckhardt, Pravesh Kothari, Madanlal Musuvathi, Santosh Nagarakatte

**Summary:**

- ***Why read this paper?*** Reading the title, I was curious to know how probabilistic modeling can give guarantees in finding bugs. As the authors point out, concurrent programs are error prone and the possibility of finding a bug in large number of schedules is incredibly difficult and to add insult to the injury, these errors may sometimes be not reproduceable. The authors provide theoretical and empirical results that proves probabilistic guarantees on several production scale concurrent programs.
- ***Observations:*** The algorithm relies on the fact that bugs are not adversarial in practice and only a few instructions execute by a small number of threads causes bugs and if PCT is able to detect and schedule these instructions correctly, then it can find bugs successfully irrespective of numerous scheduling ways possible.
- ***Contributions:*** The authors present several results. (a) finding bugs faster than previous known techniques on well tested benchmarks. (b) Finding new previously unknown bugs (c) empirically show that the algorithm works better than the theoretical worst-case bound.
- ***Algorithm:*** The proposed algorithm is tested on for already known inputs that generate concurrency bugs. Perform randomized scheduling by inserting random delays, context switches or changing thread priorities. Also, randomly assign priority to each thread at the beginning and then change these priorities after a priority change is triggered (time elapsed). Depth is defined as minimum number of constraints needed to find a concurrency bug and PCT is designed to provide guarantees for bugs with smaller depth. A preemption bound is smallest number of preemption sufficient to find a concurrency bug. In a program with complex control flow, depth of a bug might not be readily apparent.
    - ***Design of randomized scheduler:*** Each thread has an assigned priority and a thread with low priority is selected only in case when all threads with higher priorities are blocked. Threads can change priorities when the *priority change point* is crossed. The algorithm works randomly but the randomization is controlled and dependent on other random variables like depth and number of instructions which lead to concurrent bugs. This design guarantees that the algorithm can find a bug of depth d with probability at least 1/n$k^{d-1}$.
- The results of this paper are interesting and can reduce the efforts spent in stress testing.

***Discussion:***

**Advantages:**

- Randomly scheduling can lead to missing even simple bugs and the algorithm presented is very interesting.
- The explanation in the paper is very clear although I think I need more time to understand the definitions and math involved in the paper.

- The author gives clear examples when talking about a specific concurrent bug which helps understand the problem better.
- After explaining the mentioned results, they propose design choices and sync operations as optimizations to their algorithm.
- Increasing the probability of finding a bug from $1/n^k \ to \ 1/nk^{d-1}$ is a great improvement, especially if d is small.

**Disadvantages:**

- I found the paper a bit difficult to read. The algorithm was understandable but the technical part was a bit unclear probably because of my lack of knowledge in the domain.
- In the results, they show that PCT performs almost similar to stress testing when number of threads increase when started with the same seed which raises performance questions since stress testing is widely used. It would be interesting to see if there is an upper limit to this even though the authors explicitly mentioned that this algorithm works well for small depth bugs.