Exploring the Use of Learning Algorithms for Efficient Performance Profiling

Shoumik Palkar, Sahaana Suri, Peter Bailis, Matei Zaharia
Stanford DAWN Project

Presenter: Minjun Wu
UMN CSCI 8980: Machine Learning in Computer Systems, Paper Presentation, 03/2019
Find bottleneck in program

Analyzing software production:
- Python HTML parser
- Different components (subcalls):
  - start with, append, strip, split, match, find, others

Profiler:
- Signal interrupt => statistical profiler
- Instrumenting code => tracing profiler
Existing methods

Statistical Profiler:
- Missing infrequent events
- Too much for low variance events

Tracing Profiler:
- Overhead
- Instrumentation tool
User target and opportunity

User Target:
- User wants to identify bottleneck
- User doesn’t care too much about short running and low variance components

Idea:
- Measure more for longer running and high variance parts of program
- Fewer time profiling for others
Paikana: choosing function calls to profile

Paikana proposes two ways to choose:

a. A racing algorithm: by statistical result
b. Multi-armed bandits: more standard scenario

A racing algorithm:
- Choose component with minimum running time
- Have enough confidence interval
Multi-armed bandits problem

Problem description: You have **K slot machines**, and each machine provides a **random reward** from a **probability distribution** specific to that machine. The objective is to **maximize** the sum of rewards earned through a **sequence** of lever pulls.

Problem analysis: “exploration” (try new action) v.s. “exploitation” (focus on seemingly highest reward one)
Multi-armed bandits problem (cont’)

Standard solutions:
- Naive: random try for a while then focus on the best
- Thompson sampling: the best in confidence*winrate
- UCB: choose high winrate and low variance

Connection to profiling:
- Exploration: profile subfunctions
- Exploitation: profile more on user interested one
  (longer running time and high variance)
Paikana’s solution

Successive Rejects algorithm from ref [2] (COLT 2010): “First the algorithm divides the time (i.e., the n rounds) in K − 1 phases. At the end of each phase, the algorithm dismisses the arm with the lowest empirical mean. During the next phase, it pulls equally often each arm which has not been dismissed yet.”

Parkana profile all candidates equally in each phase, then remove the least valuable one from the candidates.
Result

Shown in two figures:
- Runtime overhead: similar to statistical profiling (low)
- Profiling accuracy: close to runtime profiler (high)
Discussion, my opinion

Strength:
- Combination between program profiling and multi-armed bandits problem
- Insight: users focus on bottleneck components

Weakness:
- Components probability distribution model
- Different testing scenarios, e.g. burst v.s. Stable, or bottleneck migrations (on the fly taking back)
Thanks!