Benchmarking

Goals and Strategies

Di Jin

BROWN

Basics

Realism

Use the most realistic benchmark you can get. It can (and should) also impact your system design.

- Domain-specific advantages in real-world examples.
  - Good unit tests
  - Tracing infrastructure
- Accurate representation of the problems
  - Scale
  - Access patterns
  - Bottlenecks (*e.g.*, IO, network)
- Acquisition
  - Research papers (selected benchmarks)
  - Popular open-source projects
  - Research papers (artifact)

Be Complete and Be Frank

Ultimate goal: allowing readers to extrapolate what will happen with their workload.

- Show good things
  - Most ideal scenario/usecase
  - Best workload
- Show the bad things
  - When does the scalability stop
  - What workload will be negatively impacted
- Show the performance breakdown
  - Different component's overhead scales differently
  - Future research can aim to improve specific part
  - Readers can confirm their understanding

What to Benchmark?

Extraordinary claims require extraordinary evidence.

- Something extremely surprising needs a lot of evidence
    - *e.g.*, my system uses a single core but it's faster than a multi-threaded solution
    - Is there a breakdown?
    - How does it work on different types of workload?
    - What's the CPU utilization?
    - What are the potential limits?
    - . . .
- Something apparently true may require no benchmarking
    - *e.g.*, my system adds metadata to `pip` packages for supply-chain security, it incurs no runtime memory overhead

Infrastructure

General Setup

- Consider use managed server instances (*e.g.*, CloudLab `https://www.cloudlab.us/`, AWS)
- Daemonize the benchmarking process (*e.g.*, `screen`, `tmux`, `nohup`)
- Repeatable
  - Setup correct environment
  - Avoid stale states
  - *Seriously* consider things like Docker

Anticipation for Problems

The moment all benchmarks run without any issue is probably the last time you run them. Anticipate problems, and be prepared to solve them.

- Granularity: don't put all tests into one scripts
- Fast iteration
  - Have small scale benchmarks to help catch problems early (run in seconds)
  - Have big scale benchmarks for main result (run this remotely, maybe hours or days)

Validity

"Watch it fail first"

- Be very very paranoid
- Verify whether things work as expected
  - Rough performance numbers
  - Deliberately triggered failures
  - Workload with predictably different outcomes
  - Manual inspections (*e.g.*, debuggers, logging)

Automation

- Push-button effort
  - The benchmarks will be run **a lot**.
  - If you are lucky, they will be run by other people as well.
- Separate data generation and presentation
  - Allows analysis and inspection without re-running the experiment
  - Allows experimentation with data presentation when writing the paper
- Only add abstractions as needed. Avoid over-design.

Some Points on Methodology

No Interference

Do not run other things on the test environment

- No personal laptop (GUI, notifications, daemons)
- No running web server
- No resource sharing (even when VM is appropriate)

Statistical Reporting

Report average of **repeated** experiments, and report **standard deviation** or **confidence interval**

- Not acceptable to say 5 second improvement with 20 second variation
- ~~Pick a good one~~ (You would go to academic jail for this)
- ~~Repeat 64 times to bring P-value down~~
- Figure out why it's varying so much and fix it
  - Interference
  - Networking in non-networking benchmarks
  - CPU auto-scaling/temperature throttling

Get Some Results First

Get 3–5 benchmarks end-to-end first

- Avoid fundamental problems
  - Do not go three months into a project then realize that Python's GIL is a problem
- Prioritize important things: *"If a tree falls in a forest and no one is around to hear it, does it make a sound?"*
  - Solve problems impacting the benchmarks
  - If you believe something is important, find corresponding benchmark first

Other Things

- Add additional tracing for to breakdown overhead
- Beware of the bottleneck of the system
    - *e.g.*, don't hide CPU overhead under network bottleneck
- Differentiate latency and throughput

BROWN

Further Readings

Systems Benchmarking Crimes by Gernot Heiser

`https://gernot-heiser.org/benchmarking-crimes.html`