Benchmarking

Java Microbenchmark Harness (JMH)

Apache Bench (ab)

What is a benchmark?

- “A benchmark is a standard or point of reference against which things may be compared or assessed. In the context of software, benchmarking involves running a series of tests to measure the performance of a system, application, or component under specific conditions.” – GitHub Copilot
- Do we need benchmarks? Shouldn’t we just push to Prod and update retroactively?

Reasons for benchmarking

- **Performance Evaluation**: Benchmarking helps in evaluating the performance of different systems or components, allowing developers to identify bottlenecks and optimize code for better efficiency.
- **Comparison**: It allows for the comparison of different algorithms, libraries, or hardware configurations to determine which performs best under given conditions.
- **Validation**: Benchmarking can validate that performance improvements or changes in the codebase have the desired effect and do not introduce regressions.

Benchmarking mistakes

- Unrealisting test conditions
- Overlooking performance variability
- Not learning the specifics of the framework or language you are attempting to benchmark in (see https://www.youtube.com/watch?v=EH12jHkQFQk and https://benjdd.com/languages3/)

JMH

- Useful tool for writing Java “nano/micro/milli/macro benchmarks” – https://openjdk.org/projects/code-tools/jmh/
- Common pitfalls:

1. **Ignoring Warmup**: Always include a warmup phase to ensure the JVM has optimized the code before measurements begin. This helps in getting more accurate results.
2. **Insufficient Iterations**: Ensure you have enough iterations for both warmup and measurement phases. Too few iterations can lead to unreliable results.
3. **Inconsistent State**: Make sure the state of the benchmarked code is consistent across iterations. Use the @Setup and @TearDown annotations to manage state properly.

JMH (Cont.)

1. **Benchmarking Small Methods**: Be cautious when benchmarking very small methods, as the overhead of the benchmarking framework can distort the results. Consider using larger workloads or aggregating multiple operations.
2. **Ignoring JVM Options**: Different JVM options can significantly affect performance. Make sure to use consistent JVM options across benchmarks.
3. **Not Isolating Benchmarks**: Run benchmarks in isolation to avoid interference from other processes or benchmarks running on the same machine.
4. **Ignoring GC Effects**: Garbage collection can impact benchmark results. Use JMH options to control and monitor GC behavior during benchmarks.
5. …. Any others you can think of?

Apache Bench (ab)

- Lightweight command line tool for benchmarking your Apache HTTP server - https://httpd.apache.org/docs/2.4/programs/ab.html