# Evaluating the Evaluation: A Benchmarking Checklist

30 Jun 2018

A co-worker introduced me to Craig Hanson and Pat Crain's performance mantras, which neatly summarize much of what we do in performance analysis and tuning. They are:

**Performance mantras**

1. Don't do it
2. Do it, but don't do it again
3. Do it less
4. Do it later
5. Do it when they're not looking
6. Do it concurrently
7. Do it cheaper

These have inspired me to summarize another performance activity: evaluating benchmark accuracy. Accurate benchmarking rewards engineering investment that actually improves performance, but, unfortunately, inaccurate benchmarking is more common. I have spent much of my career refuting bad benchmarks, and have developed such a knack for it that prior employers would not publish benchmarks unless they were approved by me. Because benchmarking is so important for our industry, I'd like to share with you how I do it.

Perhaps you're choosing products based on benchmark results, or building a product and publishing your own benchmarks. I recommend using the following benchmarking questions as a checklist:

**Benchmarking checklist**

1. Why not double?
2. Did it break limits?
3. Did it error?
4. Does it reproduce?
5. Does it matter?
6. Did it even happen?

In more detail:

## 1. Why not double?

If the benchmark reported 20k ops/sec, you should ask: why not 40k ops/sec? This is really asking "what's the limiter?" but in a way that motivates people to answer it. "What's the limiter?" sounds like a homework exercise of purely academic value. Finding the answer to "Why not double?" might motivate everyone to find and fix the limiter, and double the benchmark numbers! Answering this usually requires analysis using other observability tools while the benchmark is running (what I call "active benchmarking").

## 2. Did it break limits?

This is a sanity test. I've refuted many benchmarks by showing that they would require a network throughput that would far exceed the maximum network bandwidth (off by, for example, as much as 10x!). Networks, PCIe busses, CPU interconnects, memory busses, and storage devices (both throughput and IOPS), all have fixed limits. Networking is the easiest to check. In some cases, a benchmark may appear to exceed network bandwidth because it returns from a local memory cache instead of the remote target. If you test data transfers of a known size, and show their rate, you can do a simple sum to see the minimum throughput required, then compare that to network links.

### 3. Did it error?

What percentage of operations errored? 0%, 1%, 100%? Benchmarks may be misconfigured, or they may be stress testing and pushing targets to error-inducing limits. Errors perform differently from normal operations, so a high error rate will skew the benchmark. If you develop a habit of reading only the operation rate and latency numbers from a lengthy benchmark report (or you have a shell script to do this that feeds a GUI), it's easy to miss other details in the report such as the error rate.

### 4. Does it reproduce?

If you run the benchmark ten times, how consistent are the results? There may be variance (eg, due to caching or turbo boost) or perturbations (eg, system cron tasks, GC) that skew a single benchmark result. One way to check for system perturbations is via the load averages: let them reach "0.00, 0.00, 0.00" before you begin benchmarking, and if they don't get to zero, debug and find out why.

### 5. Does it matter?

If I showed that the wheel on one car would fly apart if spun at 7,000 mph, and the wheel on another car at 9,000 mph, how much would that affect your purchasing decision? Probable very little! But I see this all the time in tech: benchmarks that spin some software component at unrealistic rates, in order to claim a "win" over a competitor. Automated kitchen-sink benchmarks can also be full of these. These kinds of benchmarks are misleading. You need to ask yourself: how much does this benchmark matter in the real world?

### 6. Did it even happen?

Once, during a proof of concept, a client reported that latency was unacceptably high for the benchmark: over one second for each request! My colleague began analysis using a packet sniffer on the target, to see if packets were really taking that long to be returned by the server. He saw nothing. No packets. A firewall was blocking everything – the reported latency was the time it took the client to time out! Beware of misconfigurations where the benchmark doesn't actually run, but the benchmark client thinks it did.

I hope you find this methodology useful, and feel free to forward this to anyone doing or evaluating benchmarks. I'll add this methodology to my online collection: Performance Analysis Methodology.

---

*You can comment here, but I can't guarantee your comment will remain here forever: I might switch comment systems at some point (eg, if disqus add advertisements).*