• Today: Monitoring and benchmarking mistakes

• Purposes of monitoring
  – Tracing (follow execution)
  – Timing (measure time in various components)
  – Tuning (find frequent or most time consuming part)
  – Assertion checking
  – Coverage
    * Do you follow all paths through the code? More often used for SW testing, but powerful if you can apply it in other contexts

• What mechanisms have you used as monitors?

• Event driven vs timer driven (tracing vs. sampling)
  – Event driven: log every time certain types of events occur
  – Sampling: Wake up every once in a while to see what’s going on
  – How to decide between them?
    – gprof, top
  – Self-time vs. inherited time When to look at each?

• Strategies for avoiding overhead
  – Buffer in RAM
  – Process before sending; reduce data, compress
  – Use a different NIC
  – Use a different disk
  – Sampling
    * Data-driven
    * Random
    * Periodic
  – Batching

• What do you do when you have to drop measurements?
  – Best to know when you are actually doing so
  – Put in markers on both ends of event
- Have to be careful with bias: for example, dropping measurements about disk I/O when disk is overwhelmed

**Common mistakes in benchmarking**

- *Which ones have you made yourself?*
- Only report average behavior, not variance
- Ignore skewness - certain things get hit harder
- Controlling workload inappropriately
- * Ignoring caching - are you measuring what you think you are?
- Setting buffer sizes
- Sampling inaccuracies
- Ignore monitoring overhead
- * Not validating measurements
- * Different initial conditions
- Not measuring transient performance
- Using device utilizations for performance comparisons
- Too much data, not enough analysis

**I’d add:**

- Strawman baselines
  * Compare to something no one would ever do
  * These actually make your paper weaker, but people still do it all the time
  * Examples: Brute force search on an ordered list, random ordering/assignment, ignoring parallel processing, comparing to code compiled without optimizations turned on, etc.
  * * other examples?*
  * Justify your baselines based on previous work, common practice, or common sense

- Comparing optimized vs. non-optimized cases (tuned vs. not tuned)
  * You know your system a lot better
  * Ideally both should be set up by an expert
  * * What can you do defensively about this* (make exact environment and/or tuning parameters available)
  * * What can you do when comparing to another system?*

- Cherry-picking
  * Taking evidence (eg. specific benchmarks) that support your claim and ignoring or hiding those that don’t
  * One way out: scope your claims and make a convincing argument that the scope is important - eg. we perform better under high node churn, we perform better on a write-heavy workload, etc.
  * You are actually learning something valuable!

- Not measuring what you think you’re measuring
  * Caching just a special case
  * Making bad assumptions about where bottlenecks are
  * Making bad assumptions about what resources are being used
* Example: TCP window size / BDP, not actual available bandwidth

* How can you tell? (Intuition, check against analytical analysis, check resource usage on supposed bottleneck, look for suspicious thresholds)

  - Any other ideas?

• For next time
  
  - Read Chapter 10 (data presentation)
  
  - Finish HW #4
  
  - Keep working on the papers