1. General Feedback to All Groups

We first give feedback on things that *many* of you did and perhaps could use some feedback on. Specifically:

Overall Presentation

Make your presentation uniform! Make slides that look nice and make them all look consistent and clean. Spend time on this!

Data Presentation

Label all graph axes! You should never present a graph without clear labels on all axes – it just makes it harder to understand.

Use the right sized units! e.g., don’t put .0001 seconds when you can simply put .1 ms and don’t put 1000000 microseconds when you can put 1 second. Figure out what the best unit is for the granularity of measurement and use that.

Show all the data if you can. Not just averages – averaging hides information! Or, at least, look at all the data before you do the average, etc.

Graphs: generally, start at zero on the y-axis. Especially for bar graphs. Otherwise you could be misrepresenting the data.

Make graphs next to each other have same height/max value for y axes! This allows comparison across graphs.

Put the thing you’re comparing on the same graph near one another. Graphs facilitate comparison; thus, put the things you want the audience to compare near one another.

Use consistent colors across graphs, e.g., GRPC always blue, Thrift always green. That way, it’s easier to read graphs!

Empirical Methods

In general, you cannot meaningfully compare time across machines. Thus, when measuring, you have to think about what you can do on one machine.

A disk seek is not just an *fseek*! To get a disk to do a seek, you have to think about how to avoid caches (like the file system page cache, or the disk’s cache) and make sure the disk is read/writing blocks from far away spots on the disk.

Compare your measured numbers to spec sheets. What does the CPU/disk/SSD/DRAM expect to deliver? How do your numbers compare? Are there major discrepancies?

Testing should be done on the same setup, to facilitate comparison! Do not compare one system on one machine to another system on a completely different machine. This is not meaningful!

If the number looks weird, run it again. Ask why the number looks weird. Is it repeatable? What could be going on? What other numbers can you get to figure out what is going on? You can remove noise with careful experimentation!

Be curious! Figure out why things are the way they are. For example, a number of you used the C++ `chrono::whatever` timer. How does it work? You can find out!
Drawing Conclusions

Show data, and then try to draw conclusions. What does the data show? Make sure the reader/audience can see the same data, and draw the same conclusions from the data you have shown.

Don’t make guesses as to why - you’re probably wrong! Make a guess so as to then do subsequent measurements, and confirm/deny your guess. Having a guess is easy; showing that your reasoning is solid is hard and requires work.

Don’t draw strong conclusions when no strong conclusion is warranted! It’s better to say “I don’t know” than to put forward the wrong conclusion.

2. Feedback to Your Group

Part-1

- For the basic timer overhead, it’s better to time a lot of number of `timer()` in a loop to avoid variance.
- What does the C++ clock do? One way to figure it out is to run your program with `strace`.
- The L1 cache latency looks too fast. What’s the clock rate of your CPU? The time cannot be faster than the CPU’s cycle time.
- Branch mispredict: what you do is not branch mispredict. The program has to have a branch (e.g., `if` `else`).
  - One way is to use `quicksort` and give it a sorted array vs. not-sorted array, and save your observed time difference. Then the issue becomes how you could actually find #of mispredict. One way to do it is using Linux `perf` tool which supports monitoring HW events `branch-misses`. \(^1\) \(^2\)
  - \( \text{cost}_{\text{mispredict}} = \frac{\text{Time difference in sorting}}{\text{number of branch misses difference}} \)
- There is no disk numbers.

Part-2

- Don’t show total overhead even if the rubric tells you to! It is nonsensical.
- Lessons are obvious - but that’s ok! Use a model when you can. When designing an experiments, think about what is the target of the measurement, what’s the expected behavior, what’s the assumption you made for the basic things (e.g., your program runtime depends on your CPU speed, memory reference speed, OS network stack overhead...).
- How you choose your timeout value (1sec)? What if you change it to a very large or very small one?

Part-3

- GRPC:
  - Lots of numbers, latency, bandwidth, etc.
  - All look pretty reasonable.
  - How to measure request vs response? not same datacenter!
- Thrift:
  - GRPC faster? How can you compare across different systems?
- Others:
  - Don’t include rubrics as part of the submission on canvas. It would be great to include your slide (pdf) in the submission as well.

Handin

- Roster

---

\(^1\) `perf Examples`
\(^2\) `Linux Perf Programming`
Member | Email
--- | ---
Amola Hinge | ahinge@wisc.edu
Venkat Akhil | vboddupalli@wisc.edu
Shutao Wang | swang548@wisc.edu
Pramod Sathyanarayana | psathyanaray@wisc.edu

- Checked Handin: `amola_hinge`
- Pretty good labeling, thanks!