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

Coherent slides! overall clear presentation with good structure. There are some mysteries! But that’s normal.

Part-1

- clock_gettime(). chrono. How could you find out if the cpp chrono is calling system call? Quite simple, run your program with strace.
- It seems like main memory access is too quick! What’s the memory reference number for best-possible memory HW? You may want to avoid the CPU prefetching by force your CPU to work as you want (always do main memory reference, instead of happily accessing the L1/L2).
- Branch mispredict: not included.
  - You README has roughly the right direction.
  - 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.\footnote{Linux Perf Programming} \footnote{Linux Perf Programming}
  - $cost_{mispredict} = \frac{Time\ difference\ in\ sorting}{number\ of\ branch\ misses\ difference}$
- Disk number: put the actual number not 5000!
- Using graphs were a good idea. In Pic-1, it’s better to put L1,L2 in one graph and Mutex+memory in another because L1 cache is hard to see (too small).

Part-2

- Mystery overhead increase in the first graph! numbers otherwise make sense.
- Some small -O differences.

Part-3

- Marshalling/Unmarshalling:
  - You complex structure is not that “complex”. In the lecture of RPC paper, it was mentioned that “One thing the paper did not talk about if we really need it to work is”\footnote{perf Examples}... POINTER. So a data structure include pointers? (e.g., map/linked list...).
  - It’s a bit odd that in the slide Round Trip Latency (Same Machine) your int is way worse than double and even string.
- GRPC/Thrift:
  - Thrift generally higher except for complex objects (same machine).
  - Different machines: less difference.
  - Nice UDP comparison.
  - Grpc/etc numbers look good.
  - All on mi! one day I hope to acquire such a machine.
Weird curve shows up in *Bandwidth Comparison (GRPC, Thrift)*.
* Appears several times? then show all the data! and then the average, it will be more informative.

**Handin**

- Roster

<table>
<thead>
<tr>
<th>Member</th>
<th>Email</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preeti Nayak</td>
<td><a href="mailto:preeti_nayak@cs.wisc.edu">preeti_nayak@cs.wisc.edu</a></td>
</tr>
<tr>
<td>Shubhankit Rathore</td>
<td><a href="mailto:shubhankit@cs.wisc.edu">shubhankit@cs.wisc.edu</a></td>
</tr>
<tr>
<td>Akul Gupta</td>
<td><a href="mailto:agupta274@wisc.edu">agupta274@wisc.edu</a></td>
</tr>
<tr>
<td>Srinivas Pothuraju</td>
<td><a href="mailto:pothuraju@wisc.edu">pothuraju@wisc.edu</a></td>
</tr>
</tbody>
</table>

- Checked Handin: `akul`
- Great to see the slides are in Handin.
- Glad to see references in README.
- Add a page number into your slide will make commenting easier.
- In the UDP library, avoid using the `long totalTime` for both local variable and global variable.