Google Pro Tip: Use Back-of-the-envelope-calculations to Choose the Best Design

Wednesday, January 26, 2011 at 8:44AM

Todd Hoff in Strategy, google

How do you know which is the "best" design for a given problem? If, for example, you were given the problem of generating an image search results page of 30 thumbnails, would you load images sequentially? In parallel? Would you cache? How would you decide?

If you could harness the power of the multiverse you could try every possible option in the design space and see which worked best. But that's crazy impractical, isn't it?

Another option is to consider the order of various algorithm alternatives. As a prophet for the Golden Age of Computational Thinking, Google would definitely do this, but what else might Google do?

Use Back-of-the-envelope Calculations to Evaluate Different Designs

Jeff Dean, Head of Google's School of Infrastructure Wizardry—instrumental in many of Google's key systems: ad serving, BigTable; search, MapReduce, ProtocolBuffers—advocates evaluating different designs using **back-of-the-envelope calculations**. He gives the full story in this Stanford video presentation.

Back-of-the-envelope calculations are estimates you create using a combination of thought experiments and common performance numbers to a get a good feel for which designs will meet your requirements. Dr. Dean thinks an important skill for every software engineer is the ability to estimate the performance of alternative systems, using back of the envelope calculations, without having to build them.

He gives a great example of the process in the video, but first...

Numbers Everyone Should Know

To evaluate design alternatives you first need a good sense of how long typical operations will take. Dr. Dean gives this list:

- L1 cache reference 0.5 ns
- Branch mispredict 5 ns
- L2 cache reference 7 ns
- Mutex lock/unlock 100 ns
- Main memory reference 100 ns
- Compress 1K bytes with Zippy 10,000 ns
- Send 2K bytes over 1 Gbps network 20,000 ns
- Read 1 MB sequentially from memory 250,000 ns
- Round trip within same datacenter 500,000 ns
- Disk seek 10,000,000 ns
- Read 1 MB sequentially from network 10,000,000 ns
- Read 1 MB sequentially from disk 30,000,000 ns
- Send packet CA->Netherlands->CA 150,000,000 ns

Some things to notice:

- Notice the magnitude differences in the performance of different options.
- Datacenters are far away so it takes a long time to send anything

between them.
- Memory is fast and disks are slow.
- By using a cheap compression algorithm a lot (by a factor of 2) of network bandwidth can be saved.
- Writes are 40 times more expensive than reads.
- Global shared data is expensive. This is a fundamental limitation of distributed systems. The lock contention in shared heavily written objects kills performance as transactions become serialized and slow.
- Architect for scaling writes.
- Optimize for low write contention.
- Optimize wide. Make writes as parallel as you can.

Example: Generate Image Results Page of 30 Thumbnails

The is the example given in the video. Two design alternatives are used as design thought experiments.

Design 1 - Serial

- Read images serially. Do a disk seek. Read a 256K image and then go on to the next image.
- Performance: 30 seeks * 10 ms/seek + 30 * 256K / 30 MB /s = 560ms

Design 2 - Parallel

- Issue reads in parallel.
- Performance: 10 ms/seek + 256K read / 30 MB/s = 18ms
- There will be variance from the disk reads, so the more likely time is 30-60ms

Which design is best? It depends on you requirements, but given the back-of-the-envelope calculations you have a quick way to compare them without building them.

Now you have a framework for asking yourself other design questions and comparing different design variations:

- Does it make sense to cache single thumbnail images?
- Should you cache a whole set of images in one entry?
- Does it make sense to precompute the thumbnails?

To make these estimates realistic you'll have to know the performance of your services. If there is an unknown variable then perhaps you could rapidly prototype just that part to settle the question. To know if caching is a good design alternative, for example, you'll have to know how long it takes to write into your cache.

Lessons Learned

- Back-of-the-envelope calculations allow you to take a look at different variations.
- When designing your system, these are the kind of calculations you should be doing over and over in your head.
- Know the back of the envelope numbers for the building blocks of your system. It's not good enough to just know the generic performance numbers, you have to know how your subsystems perform. You can't make decent back-of-the-envelope calculations if you don't know what's going on.
- Monitor and measure every part of your system so you can make these sorts of projections from real data.

I personally quite like this approach. It seems much more grounded in the end-to-end nature of a system than is common. The practice today is to

focus on the trickeration of various algorithms, which are really a researchable and pluggable part of this larger, more holistic analysis.

Related Articles

- Numbers Everyone Should Know
- The Back of the Napkin by Dan Roam
- A Physicist Explains Why Parallel Universes May Exist by Brian Green

---