Google: Taming the Long Latency Tail - When More Machines Equals Worse Results

Monday, March 12, 2012 at 9:17AM

Todd Hoff in Latency, Strategy, google

> *Likewise the current belief that, in the case of artificial machines the very large and the very small are equally feasible and lasting is a manifest error. Thus, for example, a small obelisk or column or other solid figure can certainly be laid down or set up without danger of breaking, while the large ones will go to pieces under the slightest provocation, and that purely on account of their own weight. -- Galileo*

Galileo observed how things broke if they were naively scaled up. Interestingly, Google noticed a similar pattern when building larger software systems using the same techniques used to build smaller systems.

Luiz André Barroso, Distinguished Engineer at Google, talks about this fundamental property of scaling systems in his fascinating talk, Warehouse-Scale Computing: Entering the Teenage Decade. Google found the larger the scale the greater the impact of latency variability. When a request is implemented by work done in parallel, as is common with today's service oriented systems, the overall response time is dominated by the long tail distribution of the parallel operations. Every response must have a consistent and low latency or the overall operation response time will be tragically slow. The implication: high performance equals high tolerances, which means your entire system must be designed to exacting standards.

What is forcing a deeper look into latency variability is the advent of interactive

real-time computing. Responsiveness becomes key. Good average response times aren't good enough. You simply can't naively scale up techniques to build larger systems. The reason is surprising and has deep implications on how we design service dominated systems:

Google Likes Request Level Parallelism, Which is Easy

Google really likes request level parallelism, where a query is sent to many machines, but the code that runs each machine does not need to be parallelized. This is really nice and simple. To answer one query, for example, Instant Search has to instantly answer 5 or 6 queries. But it's easy to code.

Wimpy Cores Require Parallelization, Which is Hard

If you have wimpy CPUs, that is CPUs that are slow in order to be really energy efficient, then you are going to get to the point where you are going to need to parallelize each piece of code, which is harder.

It doesn’t matter if you make a CPU infinitely energy efficient if it’s responsible for on 30-40% of your budget. You can at most improve energy efficiency by 30%. There are costs in terms of engineering time and performance for going to wimpy. It's hard to code.

Scale Properties of Warehouse Scale Computing Require Reliably Low Latency

Warehouse scale computing is the term Google invented to capture their idea the **datacenter is the computer.** Warehouse-scale Computing considers computer resources to be fungible, that is they are interchangeable and location independent. Individual computers lose identity and become just a part of a service. What’s really important for Google is aggregate performance of the entire system because they do so much work in parallel.

That vision puts novel performance restrictions on your warehouse as a whole, particularly the latency tail, which Luiz illustrates with an example.

Imagine a client making a request of a single web server. Ninety-nine times out of a hundred that request will be returned within an acceptable period of time. But one time out of hundred it may not. Say the disk is slow for some reason. If you look at the distribution of latencies, most of them are small, but there's one out on the tail end that's large. That's not so bad really. All it means is one customer gets a slightly slower response every once in a while.

Lets' change the example, now instead of one server you have 100 servers and a request will require a response from all 100 servers. That changes everything about your system's responsiveness. **Suddenly the majority of queries are slow. 63% will take greater than 1 second**. That's bad.

Using the same components and scaling them results in a really unexpected outcome. This is a fundamental property of scaling systems: you need to worry not just about not latency, but tail latency, that is the longer events in your system. **High performance equals** high tolerances. At scale you can’t ignore tail latency.

A Networking Example of Latency Tail

In Microstorm in a teacup: Are you suffering from the long tail effect? there's a good discussion on the impact of long tail problems on trading:

> *A long tail distribution in a trading network can represent the distribution of latency experienced by trades*
> *The majority of trades experience very low latency, less than 5 milliseconds, but a small number can experience trade latencies upwards of 900ms. Latency can be symptomatic of a long tail anomaly such as a bandwidth saturating microburst.*

This latency could come from: RCP Library, DNS lookups, packet loss,

microbursts, deep queues, high task response latency, locking, garbage collection, OS stack issues, router/switch overhead, transiting multiple hops, or slow processing code.

For a good discussion how sensitive TCP is to packet loss and buffering, take a look at SPDY: What I Like About You.

To see how tricky these problems are to degug and correct, StackExchange had a microburst problem the was a devil to find: Per Second Measurements Don't Cut It.

The implication of all of this is that this is **real engineering**. Anyone can stand up a web server and make a decent website. Building something large is completely different in kind. It takes real skills and an incredible attention to detail at every level. Every part of system must work reliably within bounded tolerances at all times. Not everyone can make this work.

Flash has High Latency so May Not Save the World

You might expect that flash would enter and save the world. Not so fast. Luiz explains that flash is great, but it is also depressing at the same time.

Great because random reads are 5 orders of magnitude faster for RAM when compared to disk. **Flash bridges the gap** between RAM and disk, in terms of latency and bandwidth, but especially bandwidth.

Depressing because flash is really just a glorified EEPROM. You can't really write to it. You can erase whole chunks of it and begin reprogramming it. Erasing is incredibly slow. You also have to try and minimize the number of erases. This creates the kind of **performance idiosyncrasies** you don't have with disk drives.

With flash you can read 4KB of data in 100 microseconds or so, if your read is stuck behind an erase you may have wait 10s of milliseconds. That's a **100x**

increase in latency variance for that particular variance that used to be very fast.

These effects are real because they impact the latency tail, so in a strange way disk is better than flash at scale.

Disks Suck for Random IO and Flash is Good at Random IO

But flash will win in the end because flash has a **better random IO story than disk**. Disks keep getting better and better in terms of space, a little faster for sequential access, but have stayed constant for random access at 100 random reads/sec.

Disks will continue to get bigger and cheaper on a per GB basis, but that doesn't matter, because the drives are getting so big you can't actually use that extra capacity effectively.

Software will need to find a way to mitigate the problems of flash.

How Will Latency Improve?

In Paper review: warehouse-scale computing: entering the teenage decade, Andrew Wang has a good summary of the current latency situation:

> *I/O latency variability right now is terrible, with basically all durable storage displaying a long latency tail. Random accesses to spinning disks are slow, flash writes are slow, and these high-latency events muck up the latency for potentially fast events. Network I/O suffers a similar problem. Using TCP and interrupts adds orders of magnitude of latency to network requests, making fast network hardware slow again in software.*
>
> *To summarize, there are two big ideas in the talk. First,*

> *latency and variation in latency are the key performance metrics for services these days; today's web-based applications demand both to provide a good user experience. This may require reexamination of a lot of fundamental assumptions about IO. Second, increasing the utilization of resources in a cluster is important from an efficiency and performance standpoint. Server hardware should be a fungible resource that can be easily shared among different services.*

In accordance with our high tolerance theme, Luiz has an ambitious research and development agenda for squeezing out the latency in the entire stack. More on that in The Three Ages Of Google - Batch, Warehouse, Instant.

I'm really surprised that Google hasn't simply ripped all those layers out and run their abstractions directly on the hardware, used a simpler networking technology, etc. If the warehouse is the computer, why aren't they making a custom system?

Some Software Techniques

These weren't in the talk, but I found them interesting and related, so why not toss them in?

Tree of Distribution Responses

This was an article I did while ago on: Google Strategy: Tree Distribution Of Requests And Responses:

> *The idea is to create a tree of nodes. So a root node talks to a number of parent nodes and the parent nodes talk to a number of leaf nodes. Requests are pushed down the tree through the parents and only hit a subset of the leaf nodes.*

With this solution:

- **Fan-in at each level of the tree is manageable**. The CPU cost of processing requests and responses is spread out across all the parents, which reduces the CPU and network bottlenecks.
- **Response filtering and data reduction**. Ideally the parent can provide a level of response filtering so the root only sees a subset of the response data. This further reduces the network and CPU needed by the root.
- **Collocation**. The parent can be collocated with leaves on one rack, which keeps all that traffic off your datacenter networks.

Focus on the 99%

From Middleware, Fault-Tolerance and the Magical 1% A Study of Unpredictability:

> *We present an extensive empirical study of unpredictability in 16 distributed systems, ranging from simple transport protocols to fault-tolerant, middleware-based enterprise applications, and* ***we show that the inherent unpredictability in these systems arises from a magical 1% of the remote invocations***
>
> *The magical 1% suggests that, while the emergent behavior of middleware systems is not strictly predictable, enterprise applications could cope with the inherent unpredictability by* ***focusing on statistical performance indicators, such as the 99th percentile of the end-to-end latency***.

Latency Tied to Blocking Rather then Queueing

From Moving Network Server Latency Off the Disk Speed Curve:

> *We find that faster processors improve server capacity, but have little effect on latency. By experimenting with workloads of various sizes, we determine that when disk accesses occur, both mean and median latencies increase,*

though the median should be unaffected. We trace the roots of this problem to ***head-of-line blocking*** *within filesystem-related kernel queues. This behavior, in turn, causes batching and burstiness, which has little impact on throughput, but severely degrades latency. By examining individual request latencies, we find that this blocking gives rise to a phenomenon we call service inversion, where requests are served unfairly.*

Service inversion, where ***short requests are often served with much higher latencies than much larger requests****. We also find that this phenomenon increases with load, and that it is responsible for most of the growth in server latency.*

By ***addressing the blocking issues*** *both in the Apache and the Flash server, we improve latency by more than an order of magnitude, and demonstrate a qualitatively different change in the latency profiles. The resulting servers also exhibit higher capacity, lower burstiness, and more fair request handling across a wide range of workloads. Finally, our results show that most server-induced latency is tied to blocking effects, rather than queuing*

Related Articles

- On Hacker News
- Latency Is Everywhere And It Costs You Sales - How To Crush It
- Galileo on Scaling
- Measurement and Characterization of Latency in Trading Networks
- Quantiles on Streams - computing approximate quantiles over streams.
- Bitcask Rocks - for writes the 99th-percentile number mostly stays under 2ms, and only up to 6ms in the worst samples. Without tuning, both reads and writes seem to be amortizing the cost of seeks over a very large

number of operations

- You can learn a lot from a histogram by Steve Newman
- Three Latency Anomalies by Steve Newman
- Brawny cores still beat wimpy cores, most of the time by Urs Holzle
- Strategy: Facebook Tweaks To Handle 6 Time As Many Memcached Requests
-
- Characterizing Flash Memory: Anomalies, Observations, and Applications

---