AVAILABILITY AND PERFORMANCE

Feb 23, 2023

George Porter
ATTRIBUTION

• These slides are released under an Attribution-NonCommercial-ShareAlike 3.0 Unported (CC BY-NC-SA 3.0) Creative Commons license

• These slides incorporate material from:
  • Jeffrey Dean and Luiz André Barroso. The tail at scale.
Jeffrey Dean and Luiz André Barroso. The tail at scale. Communication of the ACM 56, 2 (February 2013), 74-80. DOI: https://doi.org/10.1145/2408776.2408794

Head of Google AI; (Co-)designed Google’s Ad engine, Web crawler, indexer, and query serving system. Created Spanner, BigTable, MapReduce, LevelDB, TensorFlow (AI/ML system), …

Google Fellow, VP of Engineering, Technical lead of Google’s infrastructure and datacenters
AVAILABILITY

Load Balancer

Search index servers

- CSE, Triton, Price Center
- Turing, CSE, UCSD
- Graphics, Networking, EBU3B

Search terms
AVAILABILITY METRICS

• Mean time between failures (MTBF)
• Mean time to repair (MTTR)
• Availability = (MTBF – MTTR)/MTBF

Example:
• MTBF = 10 minutes
• MTTR = 1 minute
• A = (10 – 1) / 10 = 90% availability

• Can improve availability by increasing MTBF or by reducing MTTR
• Ideally, systems never fail but much easier to test reduction in MTTR than improvement in MTBF
HARVEST AND YIELD

- \textit{yield} = \textit{queries completed/queries offered}
- In some sense more interesting than availability because it focuses on client perceptions rather than server perceptions
- If a service fails when no one was accessing it…
- \textit{harvest} = \textit{data available/complete data}
- How much of the database is reflected in each query?
- Should faults affect yield, harvest or both?
DQ PRINCIPLE

- Data per query * queries per second \(\rightarrow\) constant

- At high levels of utilization, can increase queries per second by reducing the amount of input for each response

- Adding nodes or software optimizations changes the constant
Tail Tolerance: Dependent/Sequential Pattern

- Consider iterative lookups in a service to build a web page
  - E.g., Facebook
- Issue request, get response, based on response, issue new request, etc...
- How many iterations can we issue within a deadline D?
service to feel responsive.

Variability in the latency distribution of individual components is magnified at the service level; for example, consider a system where each server typically responds in 10ms but with a 99th-percentile latency of one second. If a user request is handled on just one such server, one user request in 100 will be slow (one second). The figure here outlines how service-level latency in this hypothetical scenario is affected by very
PERFORMANCE NOT AT SCALE

• What is the expected time to service one request to one server?
  • 10ms? more? less?
What is the expected time to service three correlated requests to three servers?

- Must wait until all complete before the load balancer can return a result to the user
- 10ms? more? less?
Latency variability is magnified at the service level.
Key Observation:

- 5% servers contribute nearly 50% latency.

Why not just rid of those “slow” 5% of the servers?
FACTORS OF VARIABLE RESPONSE TIME

- **Shared Resources (Local)**
  - CPU cores
  - Processors caches
  - Memory bandwidth

- **Global Resource Sharing**
  - Network switches
  - Shared file systems

- **Daemons**
  - Scheduled Procedures
FACTORS OF VARIABLE RESPONSE TIME

• Maintenance Activities
  • Data reconstruction in distributed file systems
  • Periodic log compactions in storage systems
  • Periodic garbage collection in garbage-collected languages

• Queueing
  • Queueing in intermediate servers and network switches
FACTORS OF VARIABLE RESPONSE TIME

• Power Limits
  • Throttling due to thermal effects on CPUs

• Garbage Collection
  • Random access in solid-state storage devices
  • Twitter’s interesting take on GC...

• Energy Management
  • Power saving modes
  • Switching from inactive to active modes
RANDOM VARIABLES: NORM(0,1)
RANDOM VARIABLES: $\text{NORM}(\mu, \sigma)$
EXPLORING NORMAL RANDOM VARIABLES WITH GOOGLE SHEETS

• You too can generate observations of a normal random variable by adding this to a google sheets (or excel, numbers, etc) document:
  
  • =NORMINV(rand(),0,1)
CASE STUDY: MEMCACHED

- Popular in-memory cache
- Simple get() and put() interface
- Useful for caching popular or expensive requests
BASELINE: DATABASE-DRIVEN WEB QUERY

Web server ➔ Complex query ➔ Database ➔ Result ➔ Slow!
MEMCACHED EXAMPLE: CACHE HIT

Web server

Complex query

Result

Database

Memcached
MEMCACHED EXAMPLE: CACHE MISS

Web server

Complex query

Result

Database

Complex query

No result found!

Memcached

Store result

Slow!
CASE STUDY: MEMCACHED

- Popular in-memory cache
- Simple get() and put() interface
- Useful for caching popular or expensive requests
- LRU replacement policy

```ruby
function get_foo(foo_id)
  foo = memcached_get("foo:" . foo_id)
  return foo if defined foo

  foo = fetch_foo_from_database(foo_id)
  memcached_set("foo:" . foo_id, foo)
  return foo
end
```
MEMCACHED DATA FLOW

[Diagram showing data flow between client, M/C Server i, and Database with 'hit' and 'miss' scenarios.]
EXPERIMENT: GET/SET WITH MEMCACHED

demo code
• Consider distributed memcached cluster

• Single client issues request to S memcached servers
  • Waits until all S are returned

• Service time of a memcached server is normal w/ \( \mu = 90 \text{us} \), \( \sigma = 7 \text{us} \)
  • Roughly based on measurements from my former student
You too can generate observations of a normal random variable by adding this to a google sheets (or excel, numbers, etc) document:

Based on Memcached:

=NORMINV(rand(),90,7)
MATLAB SIMULATION

The graph shows the maximum expected latency (in us) for different simulated numbers of servers. Two distributions are considered:

- 99% N(90,50) distribution
- 50% N(90,50) distribution

The 99% distribution has a higher maximum expected latency compared to the 50% distribution.
WITHIN REQUEST SHORT-TERM ADAPTATIONS

- Tied Requests
- Hedged requests with cancellation mechanism.

<table>
<thead>
<tr>
<th>Mostly idle cluster</th>
<th>With concurrent terasort</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No hedge</td>
</tr>
<tr>
<td>50%ile</td>
<td>19ms</td>
</tr>
<tr>
<td>90%ile</td>
<td>38ms</td>
</tr>
<tr>
<td>99%ile</td>
<td>67ms</td>
</tr>
<tr>
<td>99.9%ile</td>
<td>98ms</td>
</tr>
</tbody>
</table>
REDUCING COMPONENT VARIABILITY

- Differentiating Service Classes
  - Differentiate non-interactive requests
- High Level Queuing
  - Keep low level queues short
- Reduce Head-of-line Blocking
  - Break long-running requests into a sequence of smaller requests.
- Synchronize Disruption
  - Do background activities altogether.
LARGE INFORMATION RETRIEVAL SYSTEMS

- Google search engine
  - No certain answers
- “Good Enough”
  - Google’s IR systems are tuned to occasionally respond with good-enough results when an acceptable fraction of the overall corpus has been searched.
LARGE INFORMATION RETRIEVAL SYSTEMS

• **Canary Requests**

  • Some requests exercising an untested code path may cause crashes or long delays.

  • Send requests to one or two leaf servers for testing.

  • The remaining servers are only queried if the root gets a successful response from the canary in a reasonable period of time.
HARDWARE TRENDS AND THEIR EFFECTS

- Hardware will only be more and more diverse
  - So tolerating variability through software techniques are even more important over time.

- Higher bandwidth reduces per-message overheads.
  - It further reduces the cost of tied requests (making it more likely that cancellation messages are received in time).