TAIL LATENCY AND PERFORMANCE AT SCALE

George Porter
March 7, 2019

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:
  • Dean and Barroso, CACM Feb 2013
  • Chien-Ying Chen (cchen140)
ANNOUNCEMENTS

Please read Dean and Barroso, CACM Feb 2013 (linked off course page)

Reminder: Read RAFT paper up to (but not including) section 6. I promise you that reviewing the lecture slides/podcast alone will not be sufficient to understanding how the protocol/system works.

OUTLINE

1. Quantifying performance at scale
2. The Tail at Scale
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

- \( yield = \text{queries completed}/\text{queries offered} \)
  - 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…
- \( harvest = \text{data available}/\text{complete data} \)
  - 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

**PERFORMANCE “HOCKEY STICK” GRAPH**
**GRACEFUL DEGRADATION**

- Cost-based admission control
  - Search engine denies expensive query (in terms of D)
  - Rejecting one expensive query may allow multiple cheaper ones to complete
- Priority-based admission control
  - Stock trade requests given different priority relative to, e.g., stock quotes
- Reduced data freshness
  - Reduce required data movement under load by allowing certain data to become out of date (again stock quotes or perhaps book inventory)

**MEMCACHE**

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

```plaintext
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

<table>
<thead>
<tr>
<th>Client</th>
<th>M/C Server i</th>
<th>Database</th>
</tr>
</thead>
<tbody>
<tr>
<td>opt</td>
<td>[hit]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>get(key)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>response(data)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[miss]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>get(key')</td>
<td></td>
</tr>
<tr>
<td></td>
<td>None</td>
<td></td>
</tr>
<tr>
<td></td>
<td>select * from table ...</td>
<td>[query results]</td>
</tr>
<tr>
<td></td>
<td>set(key, results))</td>
<td></td>
</tr>
</tbody>
</table>

tail tolerance: partition/aggregate

• 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
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?

OUTLINE

1. The Tail at Scale
2. Quantifying performance at scale
ANNOUNCEMENTS

Please read *Dean and Barroso, CACM Feb 2013 (linked off course page)*

Office hours moved to Wednesday 1pm

HOSTING: MULTIPLE MACHINES PER SITE

- **Problem**: Overloaded popular web site
- **Replicate** the site across multiple machines
  - Helps to handle the load
- **Want to direct client to a particular replica. Why?**
- **Balance load** across server replicas
- **Solution #1**: Manual selection by clients
  - Each replica has its own site name
  - Some Web page lists replicas (*e.g.*, by name, location), asks clients to click link to pick
HOSTING: LOAD-BALANCER APPROACH

• **Solution #2:** Single IP address, multiple machines
  • Run multiple machines behind a single IP address
  • Ensure all packets from a single TCP connection go to the same replica

HOSTING: DNS REDIRECTION APPROACH

• **Solution #3:** Multiple IP addresses, multiple machines
  • Same DNS name but different IP for each replica
    • DNS server returns IP addresses “round robin”
HOSTING: SUMMARY

- Load-balancer approach
  - No geographical diversity ✗
  - TCP connection issue ✗
  - Does not reduce network traffic ✗

- DNS redirection
  - No TCP connection issues ✔
  - Simple round-robin server selection
    - May be less responsive ✗
  - Does not reduce network traffic ✗

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

• Energy Management
  • Power saving modes
  • Switching from inactive to active modes
EFFECT OF LATENCY VARIATION

Component variability amplified by scale

- Latency variability is magnified at the service level.
REQUEST LATENCY MEASUREMENT

- Key Observation:
  - 5% servers contribute nearly 50% latency.

<table>
<thead>
<tr>
<th></th>
<th>50%ile latency</th>
<th>95%ile latency</th>
<th>99%ile latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>One random leaf finishes</td>
<td>1ms</td>
<td>5ms</td>
<td>10ms</td>
</tr>
<tr>
<td>95% of all leaf requests finish</td>
<td>12ms</td>
<td>32ms</td>
<td>70ms</td>
</tr>
<tr>
<td>100% of all leaf requests finish</td>
<td>40ms</td>
<td>87ms</td>
<td>140ms</td>
</tr>
</tbody>
</table>

KEY QUESTION

- What would you do with that 5% servers which contribute 50% latency?

  Eliminate all interactive request latencies.
  Live with

  But it’s infeasible to eliminate all variations!
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.

LIVING WITH LATENCY VARIABILITY

• Within Request Short-Term Adaptations
  • Handles latency of 10+ ms
  • Takes the advantage of redundancy

• Hedged Requests
  • Send redundant requests.
  • Use the results from whichever replica responds first.
  • The overhead can be further reduced by tagging them as lower priority than primary requests.
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>

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).
CONCLUSION

• Variability of latency exists in large-scale services.

• Live with variable request latency
  • Infeasible to eliminate variable request latency in large-scale services.

• The importance of these techniques will only increase. (scale of services getting larger)

• Techniques for living with variable latency turn out to increase system utilization