AVAILABILITY AND PERFORMANCE

Feb 24, 2022

George Porter
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.
MANAGING YOUR MENTAL HEALTH DURING CURRENT EVENTS

Putin unleashes war on Ukraine

Russian forces seize control of Chernobyl nuclear plant, Ukrainian official says

LIVE UPDATES Ukrainian President says he believes Russian ‘sabotage groups’ have entered the capital and that he is ‘marked’ as ‘target No. 1’

A remarkable Republican statement on the Ukraine Invasion

COVID-19 deaths pass 5,000 mark in San Diego County

HEALTH

BREAKING

COVID-19
Hearing postponed in case that justice law
7 minutes ago

PUBLIC SAFETY

you should be working as normal

Webcomicname.com
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
AVAILABILITY METRICS

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

Example:
- MTBF = 10 minutes
- MTTR = 1 minute
- \[ A = \frac{10 - 1}{10} = 90\% \text{ availability} \]

- 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** = 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…

- **harvest** = 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* → 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

Response time vs. System load graph with a 'Knee' indicating the point where the response time starts to increase significantly with system load.
• 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 99\textsuperscript{th}-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 higher-level queuing. Devices classes can be used uling requests for which ing over non-interactive low-level queues short policies take effect more ample, the storage ser cluster-level file-system few operations outstand erating system’s disk maintaining their own of pending disk requ
• 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.
REQUEST LATENCY MEASUREMENT

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:
  
  \[ =\text{NORMINV}(\text{rand}(),0,1) \]
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

Result

Database

Slow!
MEMCACHED EXAMPLE: CACHE HIT

Web server

Complex query

Result

Database

Memcached

Result
MEMCACHED EXAMPLE: CACHE MISS

Web server → Complex query → Database

Database → Complex query → Memcached

Memcached → Store result → Result

Result → Complex query → Web server

Slow!
CASE STUDY: MEMCACHED

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

```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

**Hit Scenario**
- Client requests "get(key)"
- M/C Server i returns "response(data)"

**Miss Scenario**
- Client requests "get(key')"
- M/C Server i returns "None"
- Client requests "select * from table ...
- M/C Server i receives [query results]
- M/C Server i sets "key, [results]"
from pymemcache.client import base

client = base.Client(('localhost', 11211))
client.set('some_key', 'some value')
print(client.get('some_key'))
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/ $\mu = 90\mu s$, $\sigma = 7\mu s$
    • Roughly based on measurements from my former student
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:

• Based on Memcached:

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

Graph showing the relationship between the simulated number of servers and the maximum expected latency (in us). The graph includes two distributions:
- 99% $N(90,50)$ distribution (dotted red line)
- 50% $N(90,50)$ distribution (solid green line)
**WITHIN REQUEST SHORT-TERM ADAPTATIONS**

- **Tied Requests**
  - Hedged requests with cancellation mechanism.

<table>
<thead>
<tr>
<th></th>
<th>Mostly idle cluster</th>
<th>With concurrent terasort</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No hedge</td>
<td>Tied request after 1ms</td>
</tr>
<tr>
<td>50%ile</td>
<td>19ms</td>
<td>16ms (-16%)</td>
</tr>
<tr>
<td>90%ile</td>
<td>38ms</td>
<td>29ms (-24%)</td>
</tr>
<tr>
<td>99%ile</td>
<td>67ms</td>
<td>42ms (-37%)</td>
</tr>
<tr>
<td>99.9%ile</td>
<td>98ms</td>
<td>61ms (-38%)</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).