CSE 124: QUANTIFYING PERFORMANCE AT SCALE AND COURSE REVIEW

George Porter
December 6, 2017
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:
  • A New Focus for a New Century: Availability and Maintainability >> Performance, Dave Patterson, USENIX Fast Keynote, Jan 2002.
ANNOUNCEMENTS

Homework 6 due this week
Project 2 due Friday at 5pm
OUTLINE

1. Quantifying performance at scale
2. Review
3. Open discussion / questions
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
• Human error largest cause of failure in the more complex service, significant in both

• Network problems largest cause of failure in the less complex service, significant in both
HARVEST AND YIELD

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

- **Y-axis**: Response time
- **X-axis**: System load

The graph illustrates a significant change in response time at the knee point, indicating a critical system load threshold.
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  • Reduce required data movement under load by allowing certain data to become out of date (again stock quotes or perhaps book inventory)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

```
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**
  - Client: `get(key)`
  - M/C Server i: `response(data)`

- **Miss**
  - Client: `get(key')`
  - M/C Server i: `None`
  - Database: `select * from table ...`
  - Client: `set(key, [results])`
1. Quantifying performance at scale
2. Review
3. Open discussion / questions
NETWORK PROGRAMMING FUNDAMENTALS

- Network sockets API: open(), connect(), send(), recv(), etc
- How names are resolved to addresses in DNS
- End to end protocols
  - Move from host-to-host to process-to-process communication model
  - TCP – provide abstraction of reliable in-order byte stream on top of IP protocol
- Signals and timeouts
- Concurrency, multi-tasking, multiplexing
- Locking, mutexes, sharing state between threads/processes
PROTOCOL DESIGN AND ANALYSIS

- Framing vs. parsing
- Delimiter vs. length-value
- Server-side protocol handling
- Request for comments documents (RFCs)
- Deep dive on HTTP
• Explain concept of 'idempotent'
• Maybe vs at least once vs at most once semantics; how to implement each of these?
• gRPC
• Role of stub compiler, RPC runtimes
• Discuss whether the following operations are idempotent:
  • Pressing a lift (elevator) request button
  • Writing data to an offset in a file
  • Appending data to the end of a file (assuming there are no other writers in the system)
REPLICA CONSISTENCY AND FAULT TOLERANCE

- Lamport clocks, vector clocks, time synchronization
- Replicated state machines/logs
- Two-phase commit
- Two-phase locking
DATA CENTERS AND CDNS

- Round-robin DNS vs load balancers. advantages and disadvantages of each
- Replication vs partitioning: advantages and disadvantages
- Terms: MTTR, MTBF, availability, yield, harvest, DQ principle
- Terms: top of rack switch, PUE, SPUE
- Tail latency vs. average latency
- Energy and power
OVERLAY NETWORKS, P2P, CHORD

• Compare and contrast aspects of flooding queries, supernodes, vs structured. Tradeoffs--which is better for joining, leaving, advertising content, querying for content.

• Assume you have a Chord system of 16 or 32 nodes with identifiers provided of the data and the servers
  • Draw the identifier circle and show which nodes the keys will be assigned to
  • Be able to perform “lookups” provided an example with finger tables provided

• Each chord node must maintain routing state.
  • Describe exactly what routing state must be maintained at each node to ensure correct function. Show what this state would be for a particular node. What is the expected lookup time of an object?
  • Describe what routing state must be maintained at each node to ensure fast lookup times. Show what this state would be for that node. What is the expected lookup time of an object?
OUTLINE

1. Quantifying performance at scale
2. Review
3. Open discussion / questions
THANK YOU FOR A GREAT COURSE!
UC San Diego