Caching and consistency

Doug Woos
Logistics notes

Problem Set 1 due tonight

Paper for Monday: Role of Distributed State
Caching

Client → Redis

Client → Redis

Client → Redis
Caching

Client

Client

Client

State machine
Caching

Cache

Cache

Cache

Cache
Today, mostly

- Client
- System + Caches

N. America

Asia

Africa
append(k, A)

read(k)

ok

append(k, B)

read(k)

ok

???

System + Caches
Terminology

Anomaly: some sequence of operations (reads and writes) that “shouldn’t” be allowed

Coherence model/consistency model/memory model

- Which anomalies are possible

- Terms more or less interchangeable

Some classes of consistency model:

- Strong consistency: matches the ideal system

- Weak consistency: could have anomalies

- Eventual consistency: anomalies are “temporary”
Why different models?

Tradeoff between:

- Performance: consistency requires sync
- Availability: want to operate when disconnected
- Programmability: weaker consistency makes applications harder to write (i.e., harder to provide app-level guarantees)

CAP

- Consistency, availability, partition tolerance
- If you want availability, must give up consistency
Strict consistency

Strict Consistency (or Linearizability)

- Equivalent to ideal model
- Reads always reflect latest write
- Concurrent operations can be executed in any order
System + Caches

append(k, A) → "ok"

read(k) → "??"

append(k, B) → "ok"

read(k) → "??"
append(k, A) → ok → read(k) → A, B
append(k, B) → ok → read(k) → A, B
append(k, A)
ok
read(k)

append(k, B)
ok
read(k)
System + Caches

append(k, A)
ok
read(k)

append(k, B)
ok
read(k)
Sequential Consistency

Sequential Consistency (or Serializability)
- Execution always equivalent to some interleaving
- Each node’s ops done in order
- An intuition: strict consistency, but without real time
System + Caches

append(k, A)
ok
read(k)

append(k, B)
ok
read(k)
System + Caches

1. `append(k, A)`
2. `read(k)`
3. `ok`
4. `A, B`
5. `read(k)`
6. `ok`
7. `append(k, B)`
8. `System + Caches`
append(k, A)
ok
read(k)
A, B
append(k, B)
ok
read(k)
A, B
append(k, B)
ok
read(k)
A, B
Weaker models

Read Your Writes + Eventual Consistency

- Facebook model, approximately
- Clients will always see their own writes
- They will eventually see everyone’s writes
- And eventually the order will be consistent
append(k, A)

ok

read(k)

A

read(k)

ok

append(k, A)

append(k, B)

ok

read(k)

B
System + Caches

append(k, A) -> ok

read(k) -> A

append(k, B) -> ok

read(k) -> B

...
Weaker models

Causal consistency

- Causal order (Lamport happens-before) observed everywhere

- Concurrent events can have arbitrary and inconsistent order

Transactional models (e.g. Snapshot reads)

- Some other consistency model + atomicity of transactions
How to implement (various types of) consistency?

“There are only two hard things in Computer Science: cache invalidation and naming things.”
— Phil Karlton

If we cache data, how do we make sure it reflects writes of other nodes?

Without sacrificing performance?
A note about processors

The Silently Shifting Semicolon (Marino 2015):

```java
Pasta p = new Pasta();
cooked = true;
if (cooked) {
    p.drain();
}
```

Why does this sometimes result in a null pointer error?
A note about processors

Processors deal with same problems
- L1 cache = 0.5ns, RAM = 100ns
- How far from the CPU is RAM in your desktop?

Architecture + compiler conspire to move semicolon

Real problem in programming languages

My take: want sequential consistency everywhere