ACID, CAP, ~~No~~ NewSQL

A 30k  overview of distributed databases

Vlad Petric

Single-system databases

Most are relational databases

A well-understood problem

- Formalized in the 90s
- Great implementations in the 2000s, including Free Software (MySQL, Postgres, etc.)
- Work really well, up to some limits

ACID + SQL

ACID

A - atomicity

C - consistency

I - isolation

D - durability

Do we really need 4 letters?

SQL

- Decent query language, based on relational algebra
- Allows non-programmers to write complex queries
  - Filtering
  - Joins
  - Aggregates
- Reasonably well-behaved - e.g., guaranteed polynomial time

Limitations of ACID SQL databases

Hard to scale beyond several systems

- Neither ACID, nor SQL scale well

However, single computer systems scaled well between 1990 and present

- Faster processors (cores), more cores
- Faster memory, higher capacity memory
- Faster storage, higher capacity storage (spinning drives, then solid state)

Then came Big Data …

We need a database to store an entire web crawl …

We need a database to for all our users' (>100M) clicks …

Ad-hoc scaling of single-system databases

First solution - ad-hoc scaling

- Partition large tables based on keys
- Distribute onto multiple servers

But:

- No real ACID across multiple nodes
- Can’t query easily across nodes
- What if a node fails? ...

Remainder of talk

Introduction

Distributed databases

- Wish list
- Fundamental limitations: CAP theorem

Real distributed database systems

Conclusions

Shameless plug

What would we like from a distributed database?

Everything from single-system database

- ACID, SQL

Scalability

- Quantity of data
- Read/Write/Complex query bandwidth should scale with the number of systems

Fault Tolerance

- System should hide node failures, network failures;

Meet the CAP theorem

CAP (Eric Brewer):

- Consistency
- Availability
- Partition Tolerance

Meet the CAP theorem

CAP:

- Consistency: reads receive the most recent value
  - Once you write something, everyone reads it

- CAP Consistency vs ACID C/I/D (Definitions matter!)

Meet the CAP theorem

CAP:

- Availability: every request receives a non-error response
  - Writes are always accepted
  - Reads see a value (doesn’t necessarily have to be the latest)

Meet the CAP theorem

CAP:

- Partition tolerance - system tolerates an arbitrary number of nodes disconnected from the rest of the system (nodes can’t talk to each other)

CAP Theorem - at most 2 out of 3

Consistent and Available => not Partition tolerant

Consistent and Partition tolerant => not Available

Available and Partition tolerant => not Consistent

CAP Theorem - in practice

In a distributed system

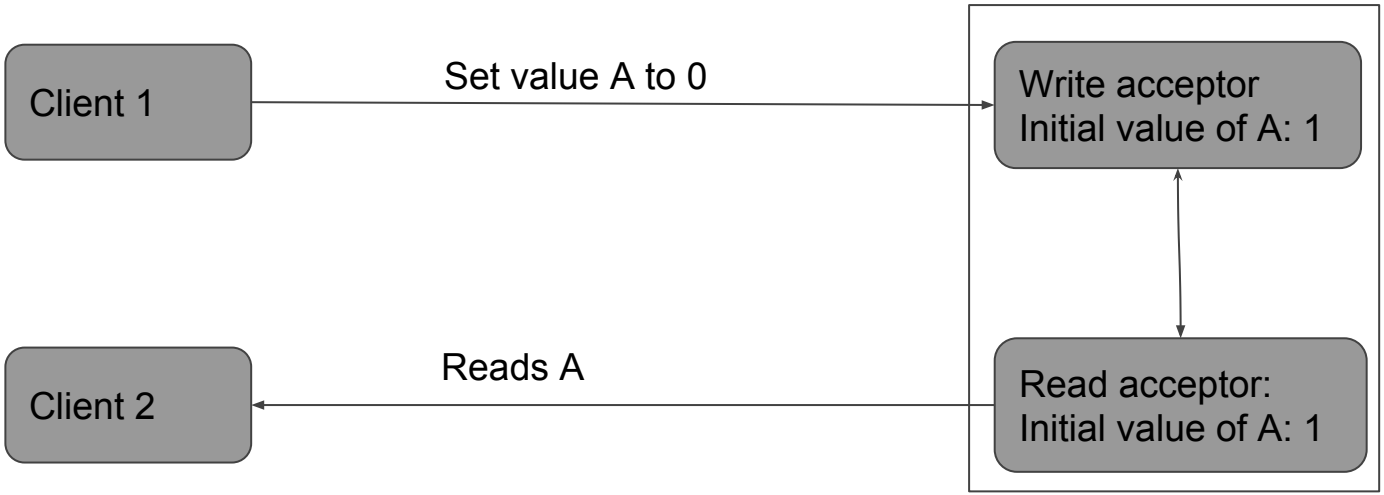
- Network issues
  - Misconfigurations
  - Power, cooling issues
- JVM stop-the-world garbage collection
- Single node failures

P happens!

When P happens - Availability or Consistency?

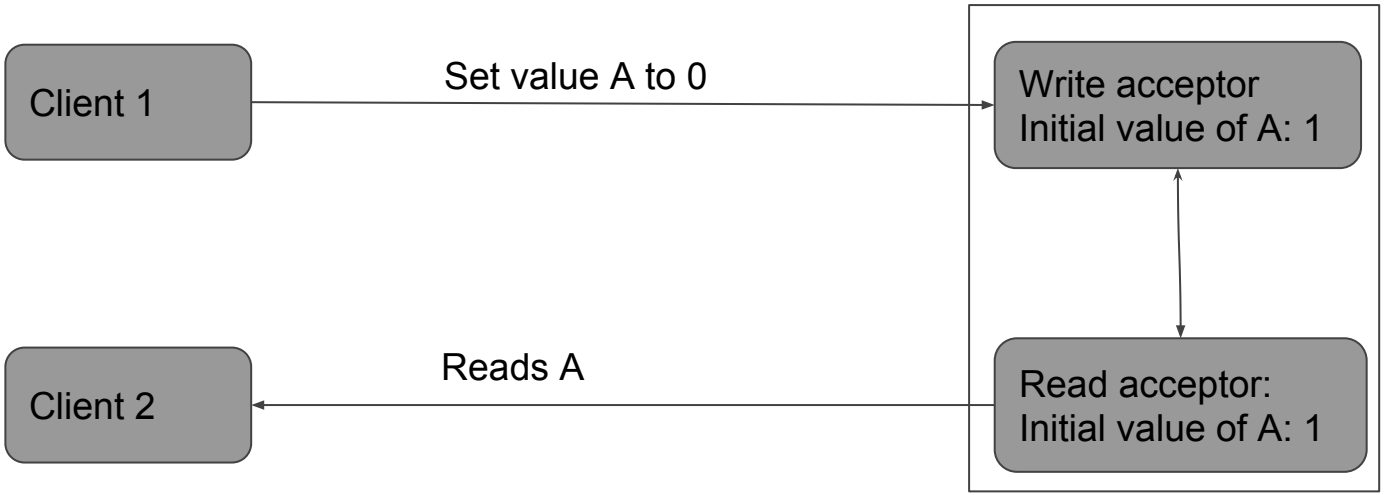
CAP Sketch of proof (1)

Distributed database; e.g. A - inventory of book “1984”



CAP Sketch of proof (1)

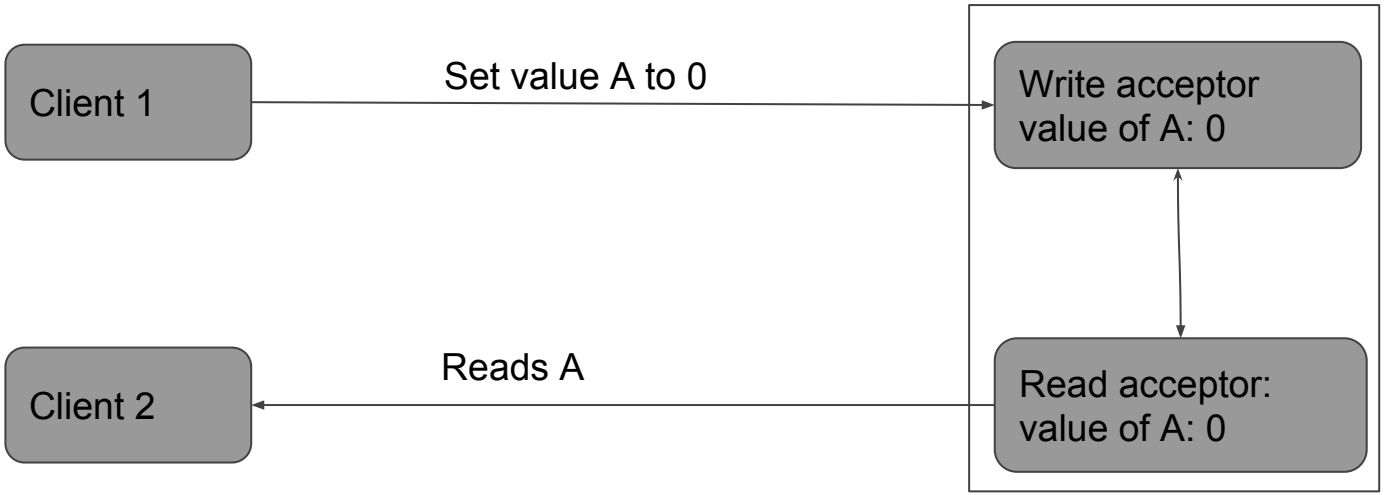
Distributed database; e.g. A - inventory of book “1984”



CAP Sketch of proof (2)

Good situation:

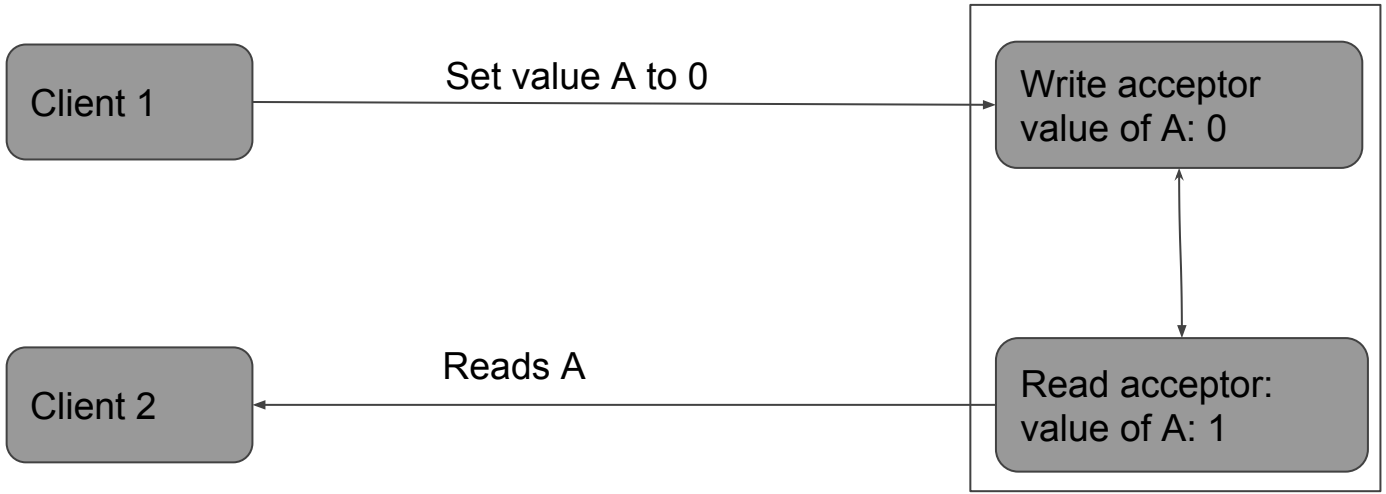
- Client 1 writes A:0
- Writer changes A:0, and propagates value to the RA
- Client 2 reads A:0



CAP Sketch of proof (3)

Inconsistent value:

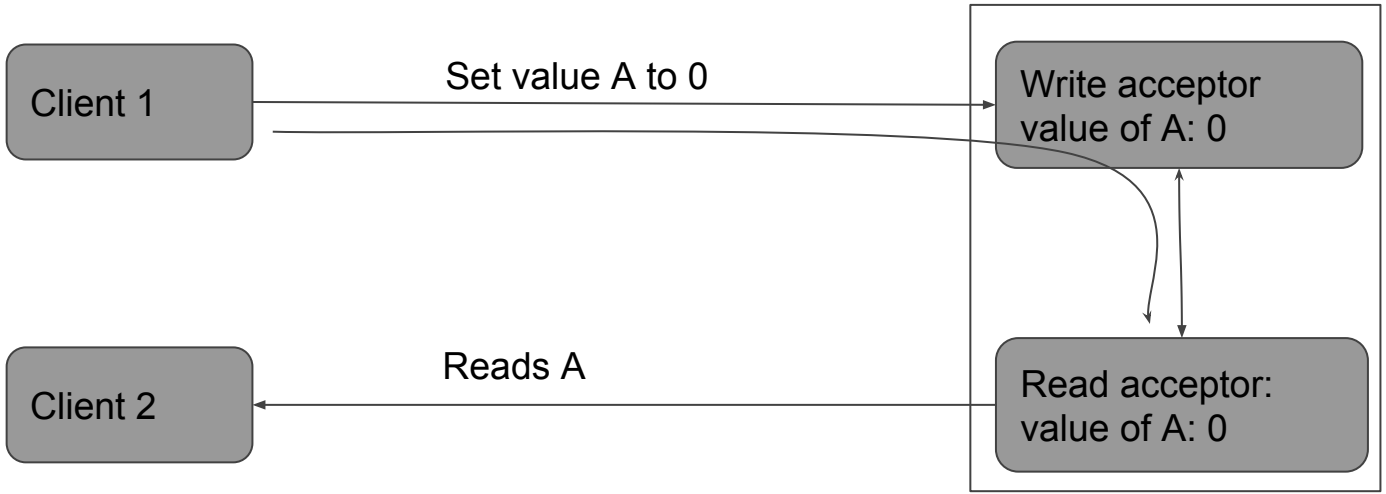
- Client 1 writes A:0
- Client 2 immediately reads A: 1



CAP Sketch of proof (4)

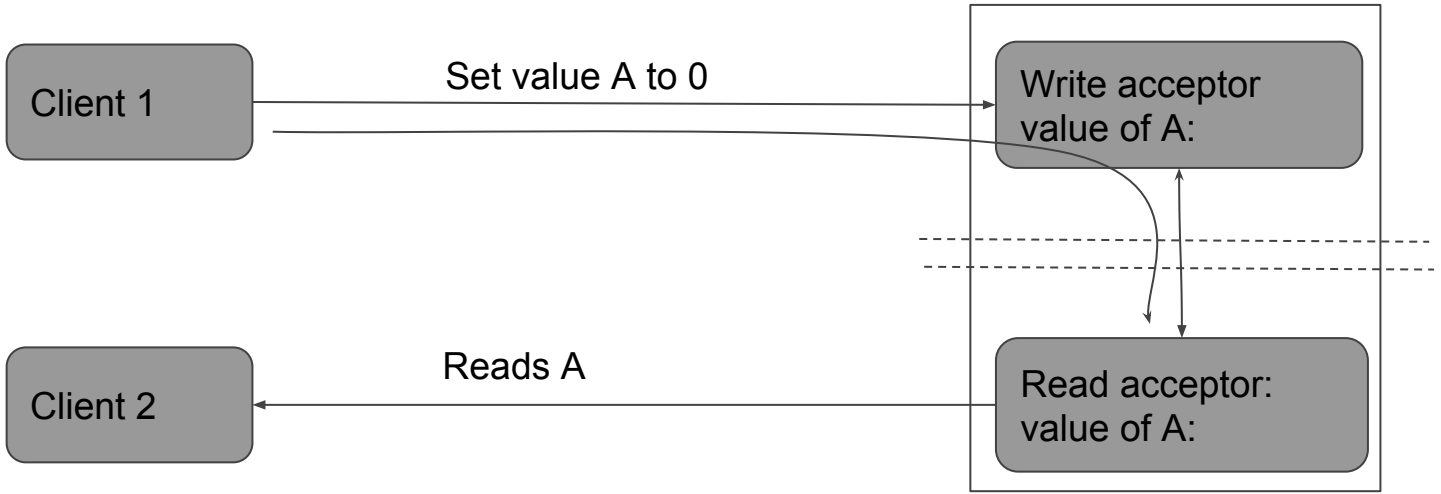
Getting consistency:

- Don’t finish the write until the read acceptor also has the most up to date value



CAP Sketch of proof (5)

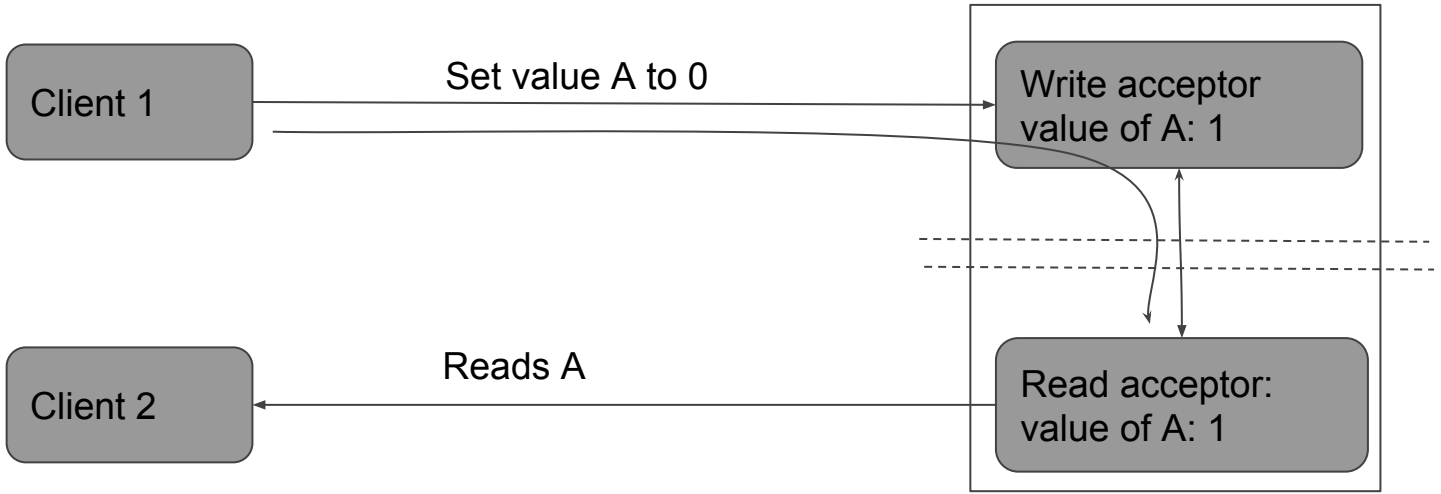
But what if the connection between W/R is severed?



CAP Sketch of proof (6)

But what if the connection between W/R is severed?

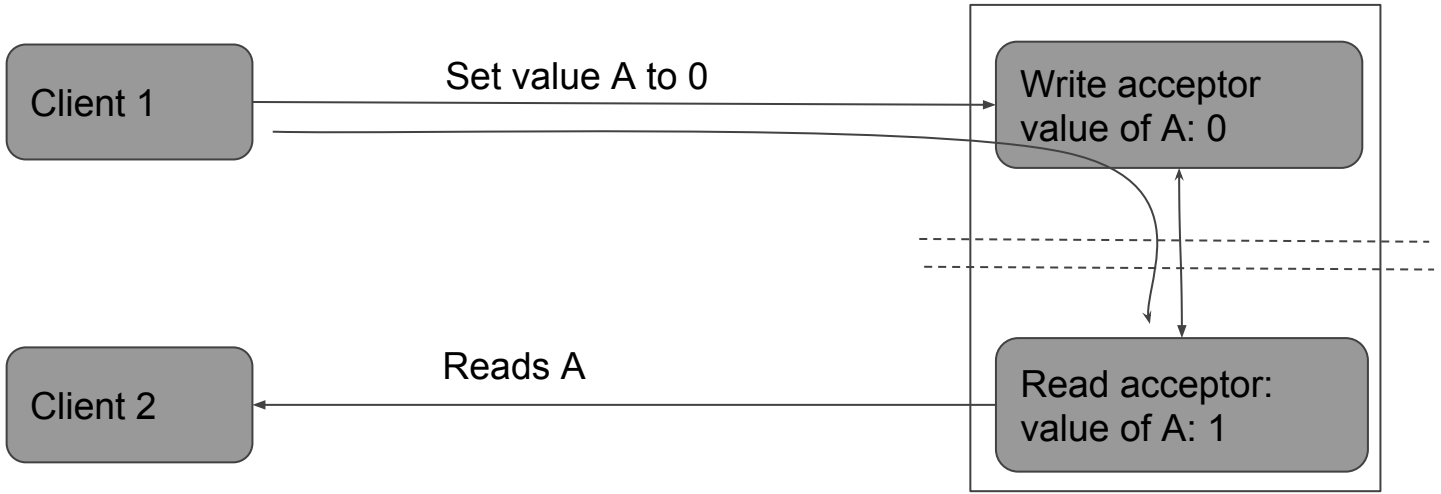
- Pretend that doesn't happen … not partition tolerant!



CAP Sketch of proof (7)

But what if the connection between W/R is severed?

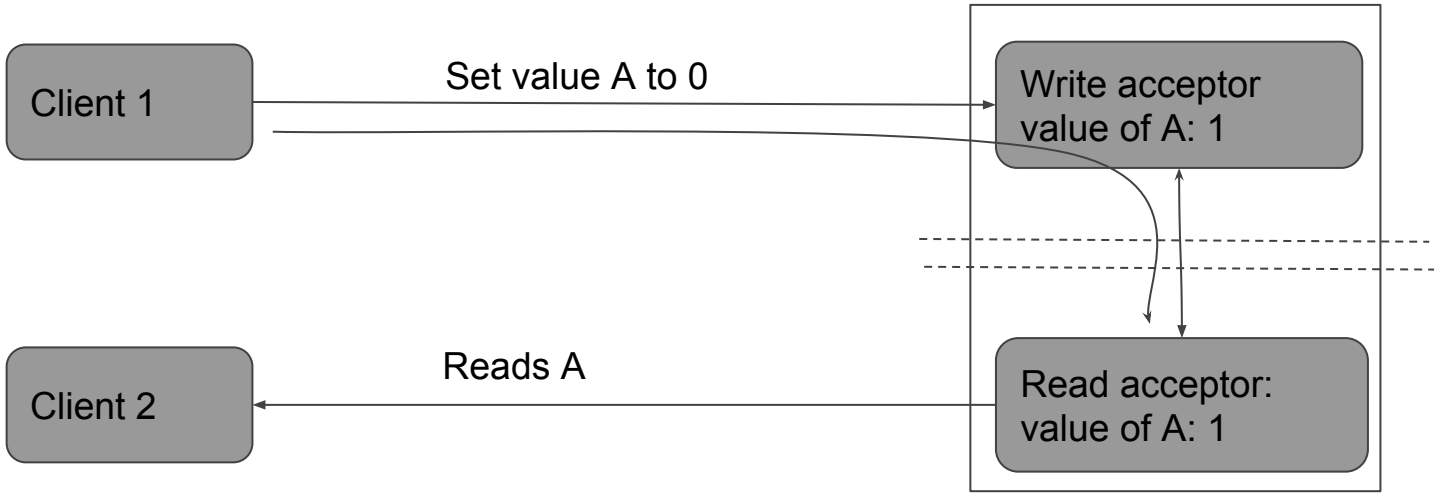
- Accept on writer, don't propagate - not consistent!



CAP Sketch of proof (8)

But what if the connection between W/R is severed?

- Fail the write - not available!



Common issues/misconceptions about CAP

- Only applies with the strict definitions of C, A, and P
  - ACID C != CAP C
  - Oftentimes you don’t need strict C
    - No need to “completely” give up C or A

- Nothing about performance (throughput, latency)

Common issues/misconceptions about CAP (2)

- No guarantee that you do get two out of the three.
  - Don’t trust the marketing department!
  - Keepin’ It Honest: http://jepsen.io/
- Even single node DB with remote clients can have CAPs.
  - Almost everything is a distributed system.

Remainder of talk

Introduction

Distributed databases

Real distributed database systems

- C trade-offs
- Existing Databases

Conclusions

Shameless plug

Consistency Levels

Strict serializability (1) - equivalent to C in CAP

Every write is seen immediately in the system.

Implies a *total ordering of operations, reads and writes*, based on time.

Consistency Levels

Serializable (2)

There is a total ordering of operations

- Execution corresponds to some serial order of operations
- … but not completely based on time.

Consistency Levels

Serializable (2)

The following writes happen in order: X, Y, Z

- Y happens after X, Z happens after Y

What will readers see? One of the following:

- Nothing
- X
- X, Y
- X, Y, Z

Consistency Levels

Eventual consistency (3)

If you stop writing, replicas converge

May not seem like much, but still offers a degree of consistency

Consistency Levels

Application-defined consistency

- If there is a conflict, the application specifies logic to handle conflicts
- Rarely used, as nobody likes to write conflict resolution logic.
- Primarily used in ad-hoc distributed databases.
- Excellent performance

Consistency Recap

Strict Serializable

Serializable

Eventually Consistent

Application-defined

Important reminder - this is a 30k feet view!

- Many variants and intermediate levels.

Strictly serializable systems

Apache Zookeeper, Google Chubby

- Distributed lock service, master election
- Look like a filesystem (hierarchical namespace)
- Can store small pieces of data as well (KiB, not GiB)
- Not suitable to high-throughput

Strict serializable systems

Zookeeper, Google Chubby

- Use N replicas
  - Every write goes to at least round up(N/2 + 1) replicas
  - Generally, odd number of replicas
- Consensus algorithm
  - Replicas agree to *the order of all writes*
- Reads:
  - For strict serializable, read from round up(N/2 + 1)
  - For serializable, read from a single replica

Strict serializable systems - partition example

---

5 replicas

P1: A, B, C split from
P2: D, E

Write to P1, P2?

Read from P1, P2?

Strict Serializable / Serializable systems

Google Spanner, CochroachDB, VoltDB

- Strict serializable as long as clocks synchronized
  - Tens of milliseconds of drift
- Serializable otherwise
- “Wait out the drift”
  - Spanner: wait on write side
  - CochroachDB: wait on read side

Strict Serializable / Serializable systems

3rd party testing was critical

- Jepsen found serialization bugs in both CochroachDB and VoltDB, subsequently fixed

CochroachDB, VoltDB - SQL subset

- Including joins!
- NoSQL became NewSQL

Eventual consistency systems

E.g., Cassandra, Big Table, Aerospike

- Any replica may accept writes.
- In case of conflict, timestamp determines who wins.
- Ordering only happens on conflict resolution

Why use eventual consistency systems?

High Throughput, Low Latency

- Easily an order of magnitude better than (strict) sequential system

High Availability of Entire system

- Not the same as CAP availability (binary property)

But … you need to be able to deal with replication delay

Some of the things I didn’t talk about

PACELC (Pass Elk) - CAP++

- Under normal operation (no P), *latency* vs *consistency*
- When P: *availability* vs *consistency*

Some of the things I didn't talk about

- What is this database suitable for?
  - Size/structure of keys/data
  - Read/Write mix
- How easy is it to manage?
  - Cassandra - easily add a replica
  - Zookeeper - need to restart whole system
- How easy is it to program?
- Special features
  - E.g., time-ordered database

Conclusions

- Definitions and Names matter
- Distributed implies complex
- Don’t trust the marketing department
- Choosing a distributed database means understanding trade-offs
- This is barely the beginning

Time for the shameless plug

About myself:

- Full name is Vlad Petric (not Vladimir), and I come from Transylvania (part of Romania).
- If you Google me, I’m not the bodybuilder
- System Architect at Cubist Systematic Strategies

Time for the shameless plug (2)

I am the author of Akro, a C++ build system with automated dependency tracking

https://github.com/vladpetric/akro

Build C++ projects with <= 3 lines of specification:

```
$COMPILE_FLAGS = "-std=c++1y" # optional

$ADDITIONAL_LINK_FLAGS = "-ltcmalloc_and_profiler" # optional

add_binary path: "homework/homework1.exe" # optional as well
```

Thank you!