Distributed Systems Theory for Mere Mortals

Ensar Basri Kahveci

Hello!

*I am **Ensar Basri Kahveci***

Distributed Systems Engineer	**@ Hazelcast**
twitter & github	**metanet**
website	**basrikahveci.com**

Hazelcast

- Leading open source Java IMDG
- Distributed Java collections, JCache, HD store, ….
- Distributed computation and messaging
- Embedded or client-server deployment
- Integration modules & cloud friendly

Hazelcast as a Distributed System

- Scale up & scale out
- Distributed from day one
- Dynamic clustering and elasticity
- Data partitioning and replication
- Fault tolerance

Distributed Systems

- Collection of entities trying to solve a common problem
  - Communication via passing messages
  - Uncertain and partial knowledge
- We need distributed systems mostly because:
  - Scalability
  - Fault tolerance

Main Difficulties

- Independent failures
- Non-negligible message transmission delays
- Unreliable communication

Systems Models

- System models come into play.
  - Interaction models
  - Failure modes
  - Notion of time
- Consensus Problem
- CAP Principle

Interaction Models

- 👊 Synchronous
- 👊👊 Partially synchronous
- 👊👊👊 Asynchronous

Hazelcast embraces the partially-synchronous model.

`OperationTimeoutException`

Failure modes

- Crash-stop
- Omission faults
- Crash-recover
- Arbitrary failures (Byzantine)

Hazelcast handles ***crash-recover*** failures by making them look like ***crash-stop*** failures.

Time & Order: Physical time

- Time is often used to order events in distributed algorithms.
- Physical timestamps
  - `LatestUpdateMapMergePolicy` in Hazelcast
- Clock drifts can break ***latest update wins***
- Google TrueTime

Time & Order: Logical Clocks

- ***Logical clocks (Lamport clocks)***
  - Local counters and communication
- Defines ***happens-before*** relationship.
  - (i.e., ***causality***)

Hazelcast extensively uses it along with ***primary-copy*** replication.

Time & Order: Vector Clocks

- Inferring ***causality*** by comparing timestamps.
- ***Vector clocks*** are used to infer ***causality***.
    - *Dynamo-style* databases use them to detect conflicts.

Lamport clocks work fine for Hazelcast because there is only a single node performing the updates.

Consensus

- The problem of having a set of processes agree on a value.
  - Leader election, state machine replication, strong consistency, distributed transactions, …
- Safety
- Liveness

FLP Result

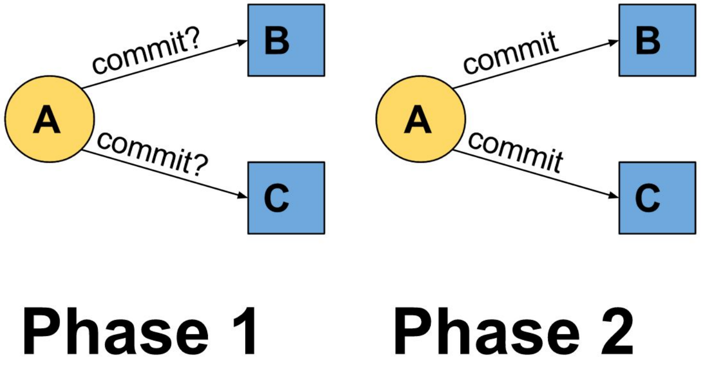
- In the **asynchronous model**, distributed consensus **may not be solved within bounded time** if at least one process can fail with ***crash-stop***.
- It is because we cannot differentiate between a crashed process or a slow process.

Unreliable Failure Detectors

- ***Local failure detectors*** which rely on ***timeouts*** and can make ***mistakes***.
- Two types of mistakes:
  - suspecting from a running process ⇒ ***ACCURACY***
  - not suspecting from a failed process ⇒ ***COMPLETENESS***
- Different types of failure detectors.

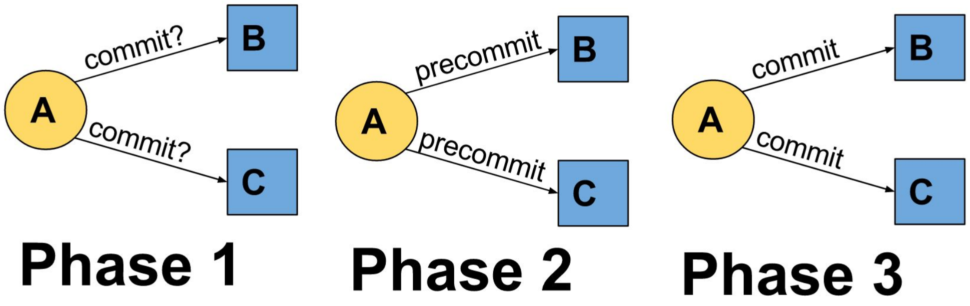
Two-Phase Commit (2PC)

- 2PC preserves ***safety***, but it can lose ***liveness*** with ***crash-stop*** failures.



Three-Phase Commit (3PC)

- 3PC tolerates ***crash-stop*** failures and preserves ***liveness***, but can lose ***safety*** with ***network partitions*** or ***crash-recover*** failures.



Majority Based Consensus

- Availability of majority is needed for ***liveness*** and ***safety***.
  - $2f + 1$ nodes tolerate failure of f nodes.
- Resiliency to ***crash-stop***, ***network partitions*** and ***crash-recover***.
- Paxos, Zab, Raft, Viewstamped Replication

Consensus: Recap

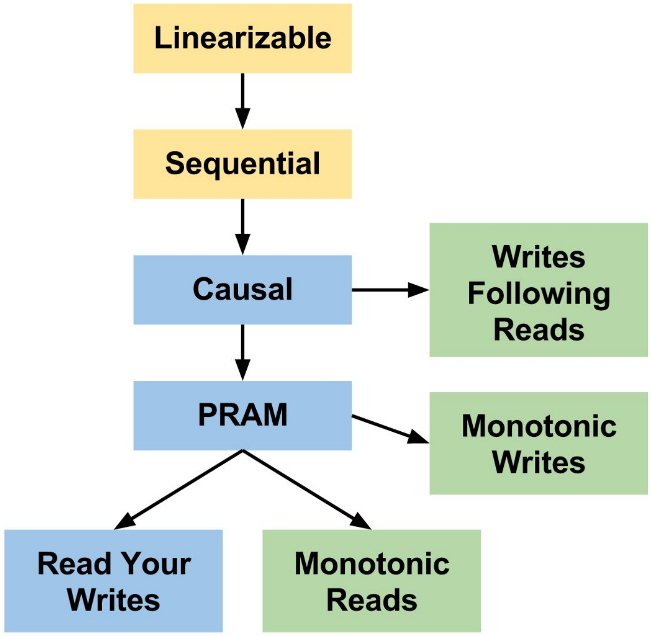
- We have 2PC and 3PC in Hazelcast, but not majority based consensus algorithms.
  - Consensus systems are mainly used for achieving strong consistency.

CAP Principle

- Proposed by Eric Brewer in 2000.
  - Formally proved in 2002.
- A shared-data system **cannot** achieve ***perfect consistency*** and ***perfect availability*** in the presence of ***network partitions***.
  - ***AP*** versus ***CP***
- Widespread acceptance, and yet a lot of criticism.

Consistency and Availability

- Levels of consistency:
  - Data-centric (CP)
  - Client-centric (AP)
- Levels of availability:
  - High availability
  - Sticky availability



Hazelcast & CAP Principle

- Hazelcast is ***AP*** with ***primary-copy & async*** replication.

	👍	👎
primary-copy	strong consistency on a stable cluster	sticky availability
async replication	high throughput	possibility of losing consistency on failures

No Hocus Pocus

- A lot of variations in the abstractions and models.
- Learn the fundamentals, the rest will change anyway.

Thanks!

Any *questions* ?