Paper Presentation on

CAP Twelve year Later: How the “Rules” Have Changed

- Eric Brewer, UC Berkeley, 2012

&

Consistency Trade-offs in Modern Distributed Database System Design

- Daniel J. Abadi, Yale University, 2012

- Anshu Maheshwari

Outline

- CAP theorem
- Why 2 of 3 in CAP theorem is misleading?
- Latency
  - The Consistency - Latency trade-off
  - CAP-Latency Connection
- PACELC – A rewrite of CAP
- Few other points about CAP

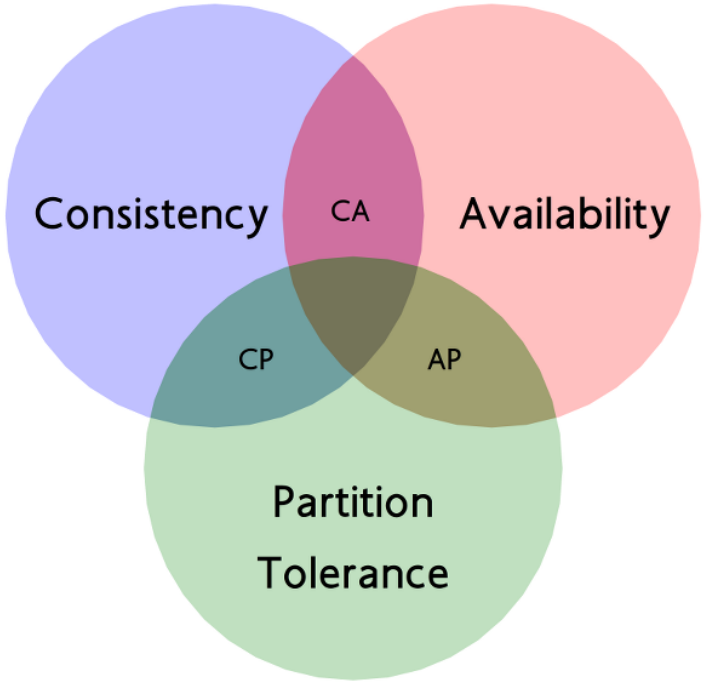
Properties of Distributed Systems

- Consistency
  - All nodes see the same data at the same time
- Availability
  - A guarantee that every request receives a response about whether it was successful or failed
- Partition tolerance
  - The system continues to operate despite arbitrary message loss or failure of part of the system

The CAP Theorem

- Any networked shared-data system can have ***at most two of the three*** CAP properties

- Eric Brewer (2000)



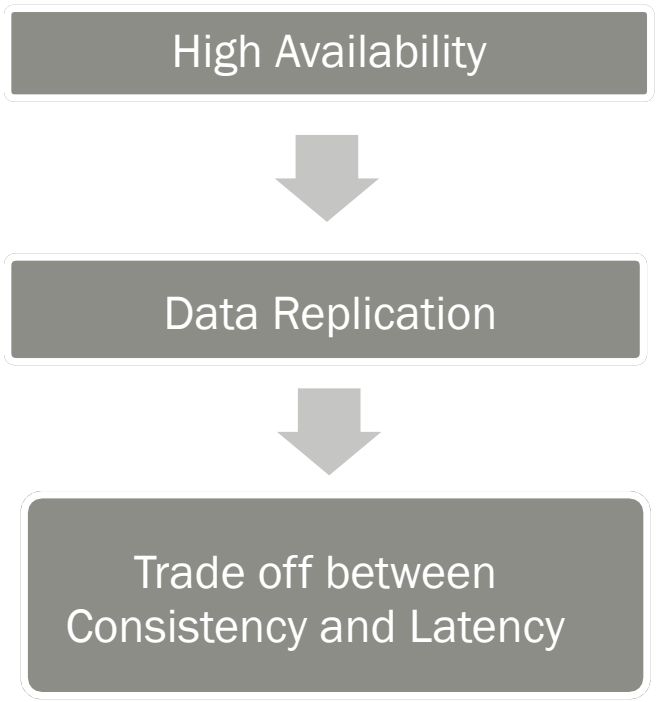
“2 of 3” is misleading! *Why*?

- Partitions are rare
  - CAP should allow perfect C and A most of the time
- When the system is partitioned, the choices between C and A can occur at granular levels -
  - Subsystem level
  - Based on operation
  - Based on user
  - Based on data etc.
- Availability is continuous (0-100%), Consistency has many levels and there can be disagreement between nodes on whether the partitions exists

Latency

“The delay from input into a system to desired outcome”

The Consistency-Latency Trade-off



- Data Replication implies a trade-off between consistency and latency as we have to update replicas
- There are three ways to send data updates –
  - Data updates sent to all replicas at the same time
  - Data updates sent to an agreed-upon location first
  - Data updates send to an arbitrary location first

Data updates sent to all replicas at the same time

- **No** pre-processing/agreement protocol
  - Result in lack of consistency
- Pre-processing/agreement protocol
  - Result in latency

Data updates sent to an master node first

- After the master nodes resolves updates, there are 3 options for replication of updated data –
  - Replication is synchronous (increase latency)
  - Replication is asynchronous
    - Systems routes all read to the master node (increase latency)
    - Any node can serve read request (lack of consistency)
  - A combination of two above – The system sends updates to some subset of replicas synchronously and rest asynchronously
    - QUORUM Protocol (a trade-off between latency and consistency)

Data updates send to an arbitrary location first

- Same as previous one, but two different updates can be initiated at two different locations simultaneously

- Again consistency/latency trade-off depending on –
  - Replication is synchronous
  - Replication is asynchronous

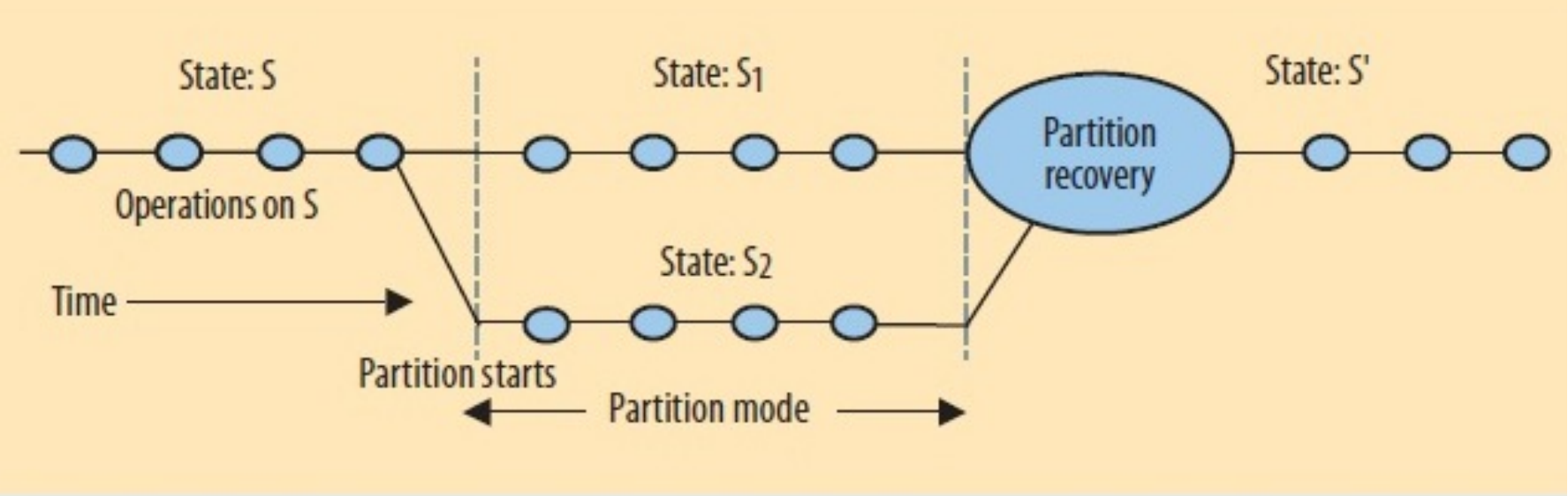
CAP-Latency Connection

- When a timeout happens, the system takes ***partition decision***–
  - Cancel the operation and decrease availability
  - Proceed with operation and risk inconsistency
- Retrying communication just delays this decision and indefinite retry is essentially C over A
- Pragmatically, a partition is a time bound on communication

Consequences

- No global notion of partition
  - Some nodes may detect partition, some may not
- Nodes that detected partition can enter partition mode
  - Optimize the consistency and availability in partition mode
- Designer can set time bounds according to their needs
  - Tighter time bounds may make subsystems enter partition mode frequently, even though the network maybe just slow and not actually partitioned

Managing Partitions



Which operations can proceed in partition mode?

- Designers need to decide about which operations can proceed, which global invariants can be violated and which cannot be

- This limits availability and consistency, e.g. –
  - Restriction on charging a credit card
  - Allowing addition of item in the cart

Partition Recovery

When the communication resumes –

- Re-enforce consistency on both sides (maintain invariants)
  - The actual procedure depends on the application –
    - Manual conflict merging (Wiki offline mode, GitHub)
    - Merge conflicts by following certain rules (Google Docs)
  - Automatic state convergence can also be done by –
    - Delaying risky operations
    - Commutative operations

- Compensate for the mistakes/violations made during partition mode
  - Issuing compensating actions - reverse transactions, refunds, coupons, charging a fee

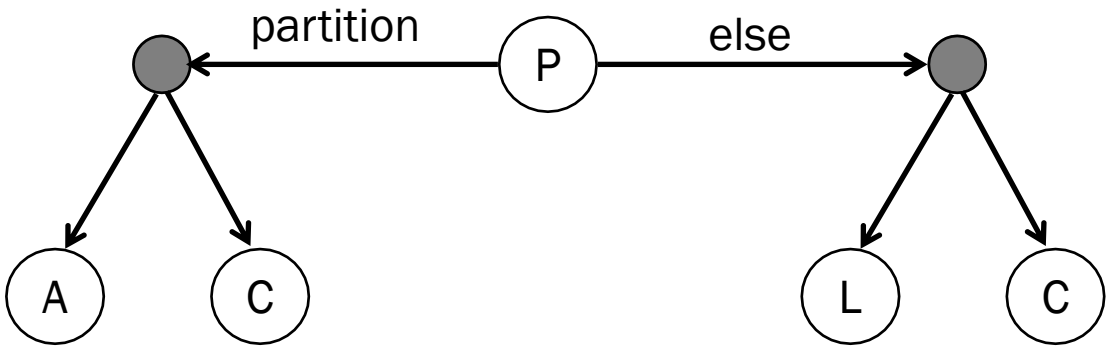
What is missing in CAP?

What are the design decisions?

- When there is no partition –
  - We need to think about the consistency and latency of the system

- When the partition happen –
  - We need a strategy to trade-off between availability and consistency

PACELC



- If there is a Partition, how does the system trade off Availability and Consistency
- Else, when the system is running in absence of partitions, how does the system trade off Latency and Consistency

Examples

- Fully ACID systems (VoltDB/H-Store) – PC/EC
- Dynamo, Cassandra – PA/EL
- MongoDB – PA/EC
- PNUTS – PC/EL

Research Problem/Future Work

- To optimize the balance between consistency and latency of a distributed database system
- Ability to access the data at granular level in Spark can reduce the latency of the operations. We can analyse the amount by which it can be reduced and the uses-cases which are most suited
- Update operations routed via Indexes in Spark can help in achieving higher level of consistency and low latency

Few other points about CAP

- Sacrificing consistency
  - When you chose A, you need to restore C in the system after recovering from the partition
  - Explicit details of all the invariants are needed

ACID and CAP

- Atomicity
  - Both side of partition should still use atomic operations
  - Higher-level atomic operations actually simplify recovery

- Consistency
  - C in ACID talks about integrity constraints
  - C in CAP talks about single-copy consistency

ACID and CAP

- Isolation
  - If system requires ACID isolation, it can operate on at most one side of partition
  - Serializability require communication, hence fails across partition

- Durability
  - During partition recovery, system may reverse the durable operations that unknowingly violated the invariant and hence need to be corrected

In general running ACID transaction on both sides of partition simplify recovery and compensation

Thank You!