Impossibility Results: CAP, PRAM, SNOW, & FLP

Network Partitions Divide Systems

How Can We Handle Partitions?

- Atomic Multicast?
- Bayou?
- Dynamo?
- Paxos?
- RAFT?
- COPS?
- Spanner?
How About This Set of Partitions?

[Diagram showing network partitions]

Fundamental Tradeoff?

• Replicas appear to be a single machine, but lose availability during a network partition

• OR

• All replicas remain available during a network partition but do not appear to be a single machine

CAP Theorem Preview

• You cannot achieve all three of:
  1. Consistency
  2. Availability
  3. Partition-Tolerance

• Partition Tolerance => Partitions Can Happen
• Availability => All Sides of Partition Continue
• Consistency => Replicas Act Like Single Machine
  • Specifically, Linearizability

Linearizability (refresher)

• All replicas execute operations in some total order

• That total order preserves the real-time ordering between operations
  • If operation A completes before operation B begins, then A is ordered before B in real-time
  • If neither A nor B completes before the other begins, then there is no real-time order
    • (But there must be some total order)
CAP Conjecture [Brewer 00]

- From keynote lecture by Eric Brewer (2000)
  - History: Eric started Inktomi, early Internet search site based around “commodity” clusters of computers
  - Using CAP to justify “BASE” model: Basically Available, Soft-state services with Eventual consistency
- Popular interpretation: 2-out-of-3
  - Consistency (Linearizability)
  - Availability
  - Partition Tolerance: Arbitrary crash/network failures

CAP Theorem [Gilbert Lynch 02]

Assume to contradict that Algorithm A provides all of CAP

```
Client 1 | Client 2
------------------------
|                      |
| w(x=1)               |
| ok                   |

Write eventually returns (from A)

Partition Possible (from P)
```

```
Client 1 | Client 2
------------------------
|                      |
| w(x=1)               |
| ok                   |

Write eventually returns (from A)

Partition Possible (from P)
```

```
Client 1 | Client 2
------------------------
|                      |
| r(x)                |
| x=0                 |

Read begins after write completes

Read eventually returns (from A)

Partition Possible (from P)
```
**CAP Theorem** [Gilbert Lynch 02]

Assume to contradict that Algorithm A provides all of CAP

- Write eventually returns (from A)
- Read begins after write completes (from A)
- Not consistent (C) \(\Rightarrow\) contradiction!

Client 1

\[ \text{w}(x=1) \]

\[ \text{ok} \]

Client 2

\[ x=0 \]

Partition Possible (from P)

**CAP Interpretation Part 1**

- Cannot “choose” no partitions
  - 2-out-of-3 interpretation doesn’t make sense
  - Instead, availability OR consistency?
- i.e., fundamental tradeoff between availability and consistency
  - When designing system must choose one or the other, both are not possible

**CAP Interpretation Part 2**

- It is a theorem, with a proof, that you understand!
- Cannot “beat” CAP Theorem
- Can engineer systems to make partitions extremely rare, however, and then just take the rare hit to availability (or consistency)

**Consistency Hierarchy**

- Strict Serializability  
  - e.g., Spanner
- Linearizability  
  - e.g., RAFT
- Causal+ Consistency  
  - e.g., Bayou
- Eventual Consistency  
  - e.g., Dynamo
Impossibility Results Useful!!!!

- Fundamental tradeoff in design space
  - **Must** make a choice
- Avoids wasting effort trying to achieve the impossible
- Tells us the best-possible systems we can build!

**PRAM** [Lipton Sandberg 88] [Attiya Welch 94]

- $d$ is the worst-case delay in the network over all pairs of processes [datacenters]
- Sequentially consistent system
  - read time + write time $\geq d$
- Fundamental tradeoff between consistency and latency!
  - (Skipping proof, see presenter notes or papers)

**Consistency Hierarchy**

- **PRAM** Theorem:
  - *Impossible* for sequentially consistent system to always provide low latency.

- **Strict Serializability**
  - e.g., Spanner
- **Linearizability**
  - e.g., RAFT
- **Sequential Consistency**
  - e.g., Bayou
- **Causal+ Consistency**
  - e.g., Bayou
- **Eventual Consistency**
  - e.g., Dynamo

- **CAP**
- **PRAM 1988 (Princeton)**
The SNOW Theorem [Lu et al. 2016]
• Focus on read-only transactions
• Are the ‘ideal’ read-only transaction possible?
  • Provide the strongest guarantees
  • AND
  • Provide the lowest possible latency?
• No 😞

The SNOW Properties
[S]trict serializability
[N]on-blocking operations
[O]ne response per read
[W]rite transactions that conflict

[S]trict Serializability
• Strongest model: real-time + total order

[Image of diagram showing permissions and transactions]

"Photo B is private!"
[S]trict Serializability

- Strongest model: real-time + total order

[O]ne Response

- One round-trip
  - No message redirection
    - Centralized components: coordinator, etc.
  - No retries
  - Save the time for extra round-trips

- One value per response
  - Less time for transmitting, marshaling, etc.

[N]on-blocking Operations

- Do not wait on external events
  - Locks, timeouts, messages, etc.

- Lower latency
  - Save the time spent blocking

[W]rite Transactions That Conflict

- Compatible with write transactions
  - Richer system model
  - Easier to program

- Spanner has W
  - COPS does not have W
The SNOW Theorem:

**Impossible** for read-only transaction algorithms to have all SNOW properties

Must choose strongest guarantees OR lowest latency for read-only transactions

**SNOW Is Tight**

- **S**: S+N+O: COPS-DW
- **N**: S+N+W: Eiger
- **O**: S+O+W: Spanner-RO
- **W**: N+O+W: Spanner-Snap

Spanner’s read-only transaction interfaces provide both sides of tradeoff!

**Consistency Hierarchy**

- **Strict Serializability**: e.g., Spanner
- **Linearizability**: e.g., RAFT
- **Sequential Consistency**: e.g., Dynamo
- **Causal+ Consistency**: e.g., Bayou
- **Eventual Consistency**: e.g., Dynamo

**Why SNOW Is Impossible** [Intuition]

Assume SNOW

W starts

A := new

B := new

W invisible

W visible

W finishes

Assume SNOW

Violates property S
“FLP”

- No deterministic 1-crash-robust consensus algorithm exists with asynchronous communication

FLP is the original impossibility result for distributed systems!

- Useful interpretation: no consensus algorithm can always reach consensus with an asynchronous network
  – Do not believe such claims!

- Led to lots and lots of theoretical work
  – (Consensus is possible when the network is reasonably well-behaved)

Conclusion

- Impossibility results tell you choices you must make in the design of your systems

- CAP: Fundamental tradeoff between availability and strong consistency (for replication)

- PRAM: Fundamental tradeoff between latency and strong consistency (for replication)

- SNOW: Fundamental tradeoff between latency and strong guarantees (for sharding)