The CAP Theorem

The story of consistency availability and partition-tolerance

Presented by: Gili Yavneh and Maya Arbel
Web services

Running On Top of Data Centers
Desirable Properties

• consistency
• availability
• partition-tolerance
Consistency

• All nodes see the same data at the same time

Data is not consistent!
Availability

• Every request receives a response
Availability

• Every request receives a response

Twilight and War and Peace
Partition Tolerance

• The system continues to operate despite network partitions.
• Example: Withdraw money even if the ATM is disconnected from the bank database.
The CAP Theorem\textsuperscript{1}

Any networked shared-data system can have at most two of three desirable properties: 

- **Partition Tolerance**
- **Availability**
- **Consistency**
Formal Model

- The network is modeled as a graph.
- It implements a read/write data object.
- Nodes communicate through messages.
- Communication is asynchronous:
  - There is no clock.
  - Nodes base decisions on messages and local computations.
Formal Model

• **Consistency:** The read/writes are *linearizable*.  
  • An operation is linearizable if it appears to the rest of the system to occur instantaneously.

• **Availability:** Every request received by a non-failing node must result in a response.  
  • Time to generate the response is not bounded.
Formal Model

- **Partition Tolerance**: The network may lose arbitrarily many messages sent from one node to another.
CAP Theorem (Formal) \(^2\)

- **Theorem:**
  It is impossible to implement a read/write data object that guarantees:
  - Consistency
  - Availability

  In all fair executions (including those in which messages are lost).
Theorem Proof

• Assume by contradiction that such a system exists.
  • Denote the system $A$.
• Assume the network contains at least two nodes.
• Divide the network into two disjoint, non-empty sets:

$G_1$ $G_2$

All messages between $G_1$ and $G_2$ are lost!
Theorem Proof

- $v_0$ – the initial value of the data object.
Theorem Proof

- $\alpha_1$ – an prefix of an execution of $A$ in which a single write of a value $v_1 \neq v_0$ occurs in $G_1$.

By availability the write completes!
Theorem Proof

• $\alpha_2$ – an prefix of an execution of $A$ in which a single read occurs in $G_2$. 
Theorem Proof

• $\alpha_2$ – an prefix of an execution of $A$ in which a single read occurs in $G_2$.
Theorem Proof

• $\alpha$ – an execution beginning with $\alpha_1$ and continuing with $\alpha_2$.
• In $G_1$ this execution is indistinguishable from $\alpha_1$.
• In $G_2$ this execution is indistinguishable from $\alpha_2$. 

Write $v_1$ to $G_1$.
Theorem Proof

• The read request began after the write finished executing.
• The execution is not linearizable—contradiction!
Corollary

• It is impossible in the asynchronous network model to implement a read/write data object that guarantees the following properties:
  • Availability, in all fair executions
  • Consistency, in fair executions in which no messages are lost
The CAP Theorem

Any networked shared-data system can have at most two of three desirable properties:

- **Partition Tolerance**
- **Availability**
- **Consistency**
A popular misconception: 2 out 3

• Can we really choose CA?
A popular misconception: 2 out 3

• Coda Hale, Yammer software engineer:³

“Of the CAP theorem’s Consistency, Availability, and Partition Tolerance, **Partition Tolerance is mandatory in distributed systems. You cannot not choose it.**”
A popular misconception: 2 out 3

• Werner Vogels, Amazon CTO:

  “An important observation is that in larger distributed-scale systems, network partitions are a given; therefore, consistency and availability cannot be achieved at the same time.”
A popular misconception: 2 out 3

• Daneil Abadi, Co-founder of Hadapt:\textsuperscript{5}

So in reality, there are only two types of systems ... I.e., if there is a partition, \textbf{does the system give up availability or consistency?}
A popular misconception: 2 out 3

• Can we really choose CA?

• CAP prohibits perfect availability and consistency in the presence of partitions, which are rare
Consistency or Availability

• There is an incredible range of flexibility for handling partitions and recovering from them
• The goal is to maximize combinations of consistency and availability that make sense for the specific application
Consistency or Availability

Normal operation mode

Detect a partition

Partition mode

Partition recovery

Compensation
CP: Best Effort Availability

• A service that guarantees consistency regardless of network behavior
• Be as responsive as possible given current network conditions
CP: Best Effort Availability

Example - Chubby

• Lock service for loosely coupled distributed systems
• Chubby operates effectively as long as no more than half the servers fail and the network is reliable
AP: Best Effort Consistency

- Users require that the application be responsive in all situations
- The response might not always be correct
AP: Best Effort Consistency
Example - Web Caches

• Services that store content such as images and video on servers in globally distributed datacenters

• Upon page request,
  • if it is available, deliver content from a nearby Web cache
  • otherwise, retrieve the data from the Web server on which the content was originally hosted

• When a webpage is updated, it might take some time for the new content to propagate
If We Lived in a Perfect World

• Assume that there are no partitions
• What is the cost of perfect consistency?
Normal Mode: Consistency vs. Latency

“Just an extra one tenth of a second in response times will cost us 1% in sales” ⁸
Normal Mode: Consistency vs. Latency

“just a half a second increase in latency caused traffic to drop by a fifth”
Normal Mode: Consistency vs. Latency

- High availability (low latency) implies that the system must replicate data

- Tradeoff between consistency and latency arises
If there is a partition (P), how does the system trade off availability and consistency (A and C); else (E), when the system is running normally in the absence of partitions, how does the system trade off latency (L) and consistency (C)?
PA/EL Systems

• Give up Consistency for availability and lower latency

• Simpler design: once a system is configured to handle inconsistencies the solution can be reused
PC/EC Systems

• Refuse to give up consistency, and will pay the availability and latency costs to achieve it
PA/EC Systems*

• In normal mode, the system guarantees reads and writes to be consistent

• If the master is partitioned, it stores all writes that have been sent to the master but not yet replicated

• Meanwhile, the system elects a new master to remain available for reads and writes

• The state of the old master and the state of the new master become inconsistent
PC/EL System

• In normal operation, give up consistency for latency
• PNUTS appears to get more consistent upon a network partition
• The system does not reduce consistency beyond the baseline consistency level when a network partition occurs—instead, it reduces availability
Back to Consistency

- All nodes see the same data at the same time
- If we give up consistency (for latency/availability) what kind of correctness do we have?
Eventual Consistency

If no *additional* updates are made to a given data item, all reads to that item will eventually return the same value
Eventual Consistency Example: Dropbox
Eventual Consistency Example: Facebook
Eventual Consistency Example: Facebook
Implementing Eventual Consistency

• Immediately respond, in the background broadcast to all others
• In case of concurrency, deterministically choose a “winning” value
• Need durability? Wait for some replicas to acknowledge the write
How Eventual is Eventual Consistency

• The system appear strongly consistent most of the time

“Our data stores returned consistent data 99.9% of the time within 13.6 ms, and on SSDs within 1.63 ms”
How Eventual is Eventual Consistency

• The system appear strongly consistent most of the time

“SimpleDB’s inconsistency window for eventually consistent reads was almost always less than 500 ms, S3’s inconsistency window lasted up to 12 seconds”
Causal Consistency

• Stronger than eventual consistency
• Operations that are causally related are seen by every node of the distributed system in the same order
Causally Related Operations

- **Execution order**: Two operations on the same node are causally ordered in the order in which they were executed.
- **Gets from**: Read operation is causally ordered after the write on the same object that stored the data retrieved by the read.
- **Transitivity**: Causal order is transitive.
Causally Related Operations Example

Execution order:
- Read A=5
- Write B=10

Gets from:
- Read B=10

Execution order and Transitivity:
- Read A=?
Causal Consistency Example: Comments

Has anyone got the new 'Reply' functionality for comments on their Page or Profile yet?

Social Safe: Well it's clearly working on this page!

Nick Bennett: and the icons get progressively smaller - multi-tier conversations - could get confusing!

Social Safe: Could be like one of those 'picture within a picture' things that goes on forever!

Glenda Snowden: No
Systems Today
Conclusions

• Choose the properties that make sense for the specific application
  • What decision will be more profitable, including possible compensation upon errors

• Remember that different parts of the system can have different properties in different times
References


References


