The CAP Theorem
Availability, Consistency, Failure in Distributed Systems
Distributed Systems

Andrea Omicini

mailto:andrea.omicini@unibo.it
Dipartimento di Informatica – Scienza e Ingegneria (DISI)
Alma Mater Studiorum – Università di Bologna

Academic Year 2025/2026

Next in Line. . .

(Hiding) Failure in Distributed Systems

- being able to keep on providing services in spite of failures is supposed to be one of the main benefit of distributed systems over centralised ones[Friedman and Birman, 1996]
- intuitive assumption: distributed systems can be designed so that if one component of the system fails – or, it becomes disconnected / partitioned – other components can replace it so as to hide failures from the outside world
    - or at least to reduce the (perceived) impact of failure

Failure & Availability

- when a system can hide (most of) it failures, it is basically (almost) *always working*
  - we say it is *highly available*
- basic questions in the design of distributed systems: *how much failure* can a given system sustain *before failure is noticed*?

Features

What do we expect from a (distributed) systems?[Gilbert and Lynch, 2012]

- we would like them to behave *correctly*, to give correct replies when queried
  - to be consistent
- we would like them to work, to make good things happen, to be *live*
  - to be available
- however, we know systems *can* actually experience power losses, crashes, network failures, message loss, malicious attacks, Byzantine failures, and so on
  - and be unreliable

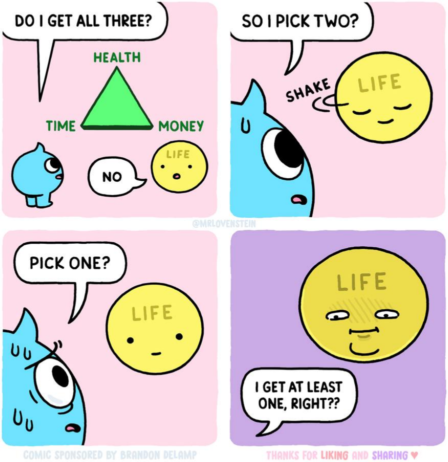
Tradeoff

The idea

- "The CAP theorem is one example of a more general tradeoff between *safety* and *liveness* in *unreliable* systems" [Gilbert and Lynch, 2012]

Next in Line. . .

Informally I



© Mr. Lovenstein

Informally II

In short[Shim, 2012]

According to the CAP theorem, it is only possible to simultaneously provide any two of the three following properties in distributed applications: consistency (C), availability (A), and partition tolerance (P).

Informally III

Less short[Zhao, 2014]

The CAP theorem [...] states that it is impossible to satisfy all three of the following guarantees:

- **Consistency** (C) the replicated data is always consistent with each other
- **Availability** (A) the data is highly available to the users
- **Partition tolerance** (P): the system can continue providing services to its users even when the network partitions

Original Theorem

First formulation by Eric A. Brewer [Brewer, 2000]

A distributed database potentially features three desirable properties

1. *C*onsistency
2. *A*vailability
3. tolerance towards network *P*artition

According to the CAP theorem, any shared-data distributed system can have at most two of these three properties

http://apice.unibo.it/xwiki/bin/view/Talk/BrewerPodc2000

An Impossibility Result

- “we repent and renounce”

? . . . yet, what is actually our sin?

- distributed systems, that’s what it is.

Pick Up One... I

... and live with(out) that

- one might *forget about tolerance to network partition*, so to have consistency and availability
- one might *forget about consistency*, and live with a partition-tolerant and highly-available system
- one might *forget about availability*, and enjoy a system that is both consistent and tolerant to network partition

Pick Up One... II

Note

- the three properties are not exactly of the same sort, both technically and conceptually
  - consistency and availability range over a spectrum of options, whereas partition tolerance can somehow be seen more as an on/off feature
- all of them are *desirable*, yet forfeiting partition tolerance is *not really an option* in real-world systems
- particularly with pervasive systems in the IoT era, where *instability* (of the network) rules[Grimm et al., 2004]

→ long story short, CAP theorem most often forces us to *choose between availability and consistency*

Example: Location-based Games I

Location-based games

- *location-based services*[Shekhar et al., 2016] use information about the position of a user / device to provide information, entertainment, security
- *location-based games* use player's location to evolve and progress the gameplay
  - e.g., BotFighters, GeoZombie[Prandi et al., 2016], Ingress, Pokémon Go, Harry Potter: Wizards Unite, Minecraft Earth

Example: Location-based Games II

CAP in location-based games: partition tolerance

- since the architecture is built around the notion of millions of players rooming physical space *worldwide* with their mobile devices (and, playing through them), forfeiting partition tolerance is *not* an option
  - mobile devices are inherently *unstable* in their network connectivity
  - players are inherently *mobile*, and so they can move in and out of network coverage
  - players tend to concentrate in some areas, e.g., during special events
  - scalabilty is a multi-level issue: not just the number of players overall, but also the number of places they can be at, and the number of players at each place
    - e.g., costly disaster at the Chicago 2017 Pokémon Go Fest

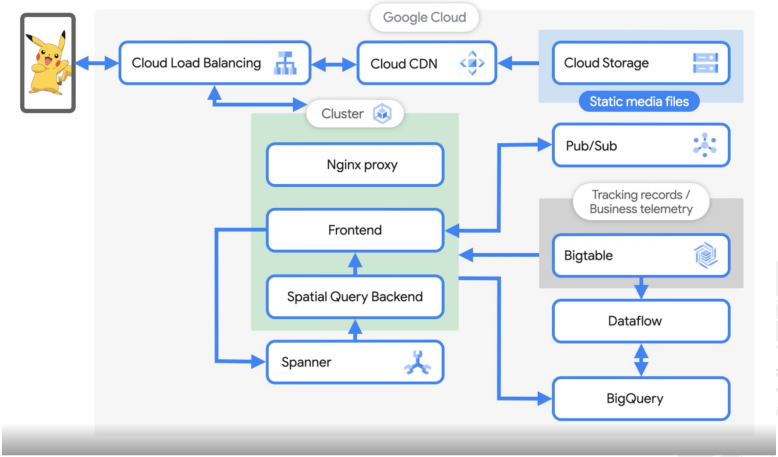
Example: Location-based Games III

CAP in location-based games: consistency and availability

- consistency of the game data (goals, situation, achievements, . . .) is essential to keep players going (and, into the game)
  - location-based games are usually built around logically-centralised architecture
  - using spatial replication in order to reduce user-perceived latency and improve scalabilty
  - when strong consistency is required, e.g., for in-game transactions, players might be forced to wait for the app servers to confirm the operation
    - by reaching a consistent state overall
- first (and usual) casualty: availability
  - the Spinning Wheel of Death 

Next in Line. . .

Catching a Pokémon on Google Cloud I



https://cloud.google.com/blog/topics/developers-practitioners/how-pokémon-go-scales-millions-requests

Catching a Pokémon on Google Cloud II

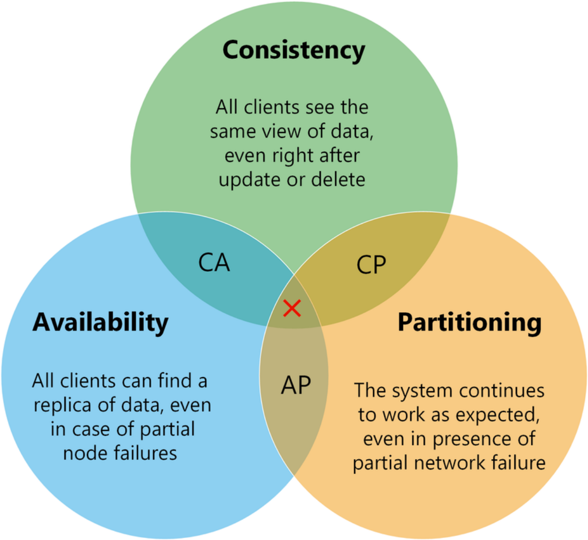
1. when a player opens the Pokémon Go app, all static media are downloaded to his/her personal device
   - which are stored in Cloud Storage
   - Cloud CDN (Content Network Delivery) uses Google's global *edge network* to serve content closer to users
   - Cloud CDN works with the Cloud Load Balancing (actually, a load balancer) to *cache* and serve user content
2. user requests are sent to NGINX reverse proxy
   - which sends this traffic to the Frontend game service
   - hosted on Google Kubernetes Engine (GKE)
3. the Spatial Query Backend handle the location-based features
   - keeps a cache *sharded* by location
   - the cache and service decides which Pokémon is shown on the map, what gyms and PokéStops are around, which is the time zone, . . .

Catching a Pokémon on Google Cloud III

4. when a player catches a Pokémon, the Frontend (GKE) sends an event to Google Spanner
   - when the write request is completed, a response is sent back to the device
   - Spanner is *strongly consistent*
5. every player's actions is recorded in the Bigtable NoSQL database service
   - as a Protobuf representation
   - for logging and tracking data
   - also as a message to a Pub/Sub topic for the analysis pipeline
6. multiple players in the same geographical region are kept in sync by *determinism* of the Pokémon Go maps
   - even if multiple players are on different machines, but in the same physical location, all players' inputs would be the same
7. *all the servers are in sync with settings changes and event timings in order for all players to feel like they are part of a shared world*

Next in Line. . .

One Way to See CAP



From Brewer's Conjecture to CAP Theorem

Going formal

- Brewer's first results required some *proof*, possibly formal
- for a proof, a more precise formulation of the conjecture is required
- a proof was soon provided [Gilbert and Lynch, 2002]
  - with some limitations
- then, several others followed
- first step for us to sketch a proof: defining the main concepts

Consistency, Availability, Network Partition I

Availability[Gilbert and Lynch, 2002]

- "For a distributed system to be continuously available, every request received by a non-failing node in the system must result in a response"
- if the system is available, we got *responses*

Consistency, Availability, Network Partition II

Consistency [Gilbert and Lynch, 2002]

- a consistent service is modelled as an *atomic data object*
- where
  - operations are totally ordered, and
  - each operation occurs in a single instant of time

! here, the meaning of *consistent* is unlike ACID properties, since it encompasses both A and C there

- among the many consequences, consistency implies that all *read* operations over a distributed shared memory occurring after a *write* operation completes must return the value of either this write operation or a later one
- if the system is consistent, we got *correct responses*

Consistency, Availability, Network Partition III

Network partition [Gilbert and Lynch, 2002]

- "When a network is partitioned, all messages sent from nodes in one component of the partition to nodes in another component are lost"
- "And, any pattern of message loss can be modelled as a temporary partition separating the communicating nodes at the exact instant the message is lost"

In short, and roughly...

- if node A send a message to node B, the network is *partitioned* if the message does not make it to B
- *availability* is when B receives the message and responds
- *consistency* is when B response is correct

Theorem I

Three different sorts of systems

The proof is given for three sorts of network

- asynchronous network with message loss
- asynchronous network without message loss
- partially synchronous network with local clocks

For teaching purposes, we focus here on the *asynchronous model*, where

- there is no single clock
- nodes act based on local computation and message received

Theorem II

Theorem [Gilbert and Lynch, 2002]

"It is impossible in the asynchronous network model to implement a read/write data object that guarantees the following properties:

- availability
- atomic consistency

in all fair executions (including those in which messages are lost)"

Proof[Gilbert and Lynch, 2002] I

Assumptions

- atomicity, availability, and partition tolerance are all fulfilled
  - *proof by contradiction* attacks just that
- nodes in the network can be partitioned into two disjoint, non-empty sets G_1, G_2
- atomic object o has initial value v_0
  - o is expected to be consistent through G_1, G_2
- α_1 executes a single write $v_1 \neq v_0$ of o in G_1
  - α_1 is the only request in that time
  - during α_1 no message are received—from G_1 to G_2, from G_2 to G_1
- no messages from G_1 are received in G_2
- α_2 executes a single read of o in G_2
  - during α_2 no message are received—from G_1 to G_2, from G_2 to G_1

Proof[Gilbert and Lynch, 2002] II

Q.E.D.

- due to availability, α_1 completes ($v_0 \rightarrow v_1$), and α_2 does, too
- executing α_1 and α_2, G_2 just sees α_2, and must return v_0
- this obviously violates consistency as defined above
  - which is basically a sort of atomic consistency

Next in Line...

What do we Make out of an Impossibility Result?

Should we stop distributing computational systems?

- in the same way we stopped using axiomatic systems after Gödel?[Gödel, 1931]
  - of course we did *not*

! negative results just *set the boundaries*, and make us understand the very *limits of our reach*

- so that an impossibility result from computer science becomes a leverage for specific and effective computer engineering methods and practices

Switching Strategy based on Partition Tolerance

A very simple scheme

- Brewer himself later elaborated on his CAP theorem [Brewer, 2012]
  (i) when the network is partitioned, a distributed system should choose a tradeoff between consistency and availability
  (ii) when there is no partition, a system can feature both consistency and availability

ACID vs. BASE I

Beyond ACID [Fox et al., 1997]

- ACID (Atomicity, Consistency, Isolation, and Durability) semantics might be too strong
  - "The design space for network services can be partitioned according to the data semantics that each service demands. "
  - "For other Internet services, however, the primary value to the user is not necessarily strong consistency or durability, but rather high availability of data"

ACID vs. BASE II

Towards BASE[Fox et al., 1997]

- "Stale data can be temporarily tolerated as long as all copies of data *eventually* reach *consistency* after a short time"
- "*Soft state*, which can be generated at the expense of additional computation or file I/O, is exploited to improve performance; data is not durable"
- "*Approximate answers* (based on stale data or incomplete soft state) delivered quickly may be more valuable than exact answers"

BASE

- Basically Available
- Soft state
- Eventual consistency

Beyond BASE

Data consistency for Cloud [Birman et al., 2012]

- most cloud services nowadays adopts BASE
  - e.g. eBay, Amazon DynamoDB
    - ! Amazon S3 allows for *strong read-after-write* consistency
    - ! Amazon DynamoDB writes are eventually consistent, reads too—but strongly-consistent reads can be configured (and, they are more costly)
- and try to mask inconsistencies from users
- steps ahead are leading to new models, based on, e.g., new data-consistency models

! further details in the literature—at the end of the slides

Overall

CAP today[Brewer, 2012]

The modern CAP goal should be to maximise combinations of consistency and availability that make sense for the specific application. Such an approach incorporates plans for operation during a partition and for recovery afterward, thus helping designers think about CAP beyond its historically perceived limitations

Next in Line...

Lessons Learnt

- in the design and development of distributed systems, we have typically to choose between a responsive system and a fully-consistent one, when network fails
- in any case, we can leverage on the CAP theorem to design your system with different features depending on the overall system status
- ACID is generally a thing to know and understand, yet less stringent models could be both theoretically and practically useful in real-world distributed systems—e.g., BASE

The CAP Theorem
Availability, Consistency, Failure in Distributed Systems
Distributed Systems

Andrea Omicini

mailto:andrea.omicini@unibo.it
Dipartimento di Informatica – Scienza e Ingegneria (DISI)
Alma Mater Studiorum – Università di Bologna

Academic Year 2025/2026

References I

[Birman et al., 2012] Birman, K., Freedman, D., Huang, Q., and Dowell, P. (2012).
Overcoming CAP with consistent soft-state replication.
Computer, 45(2):50–58
DOI:10.1109/MC.2011.387

[Brewer, 2012] Brewer, E. (2012).
CAP twelve years later: How the "rules" have changed.
Computer, 45(2):23–29
DOI:10.1109/MC.2012.37

[Brewer, 2000] Brewer, E. A. (2000).
Towards robust distributed systems (abstract).
In *19th Annual ACM Symposium on Principles of Distributed Computing (PODC '00)*, page 7, New York, New York, USA. ACM Press
DOI:10.1145/343477.343502

[Fox et al., 1997] Fox, A., Gribble, S. D., Chawathe, Y., Brewer, E. A., and Gauthier, P. (1997).
Cluster-based scalable network services.
ACM SIGOPS Operating Systems Review, 31(5):78–91
DOI:10.1145/269005.266662

[Friedman and Birman, 1996] Friedman, R. and Birman, K. (1996).
Trading consistency for availability in distributed systems.
Technical report, Cornell University
(APICe) `https://hdl.handle.net/1813/7235`

References II

[Gilbert and Lynch, 2002] Gilbert, S. and Lynch, N. (2002).
Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services.
ACM SIGACT News, 33(2):51
DOI:10.1145/564585.564601

[Gilbert and Lynch, 2012] Gilbert, S. and Lynch, N. (2012).
Perspectives on the CAP theorem.
Computer, 45(2):30–36
DOI:10.1109/MC.2011.389

[Gödel, 1931] Gödel, K. (1931).
Über formal unentscheidbare sätze der Principia Mathematica und verwandter Systeme I.
Monatshefte für Mathematik und Physik, 38(1):173–198
DOI:10.1007/BF01700692

[Grimm et al., 2004] Grimm, R., Davis, J., Lemar, E., Macbeth, A., Swanson, S., Anderson, T., Bershad, B., Borriello, G., Gribble, S., and Wetherall, D. (2004).
System support for pervasive applications.
ACM Transactions on Computer Systems, 22(4):421–486
DOI:10.1145/1035582.1035584

[Prandi et al., 2016] Prandi, C., Roccetti, M., Salomoni, P., Nisi, V., and Nunes, N. J. (2016).
Fighting exclusion: a multimedia mobile app with zombies and maps as a medium for civic engagement and design.
Multimedia Tools and Applications, pages 1–29
DOI:10.1007/s11042-016-3780-9

References III

[Shekhar et al., 2016] Shekhar, S., Feiner, S. K., and Aref, W. G. (2016).
Spatial computing.
Communications of the ACM, 59(1):72–81
DOI:10.1145/2756547

[Shim, 2012] Shim, S. S. (2012).
Guest editor's introduction: The CAP theorem's growing impact.
Computer, 45(2):21–22
DOI:10.1109/MC.2012.54

[Zhao, 2014] Zhao, W. (2014).
Building Dependable Distributed Systems.
Wiley
DOI:10.1002/9781118912744