The Cloud Trade Off

IBM Haifa Research
Storage Systems

Fundamental Requirements form Cloud Storage Systems

- ***The Google File System first design consideration:***

  "**component failures are the norm rather than the exception… We have seen problems caused by application bugs, operating system bugs, human errors, and the failures of disks, memory, connectors, networking, and power supplies. Therefore,** constant monitoring, error detection, fault tolerance, and automatic recovery **must be integral to the system**"

- ***Amazon Dymano Key-Value Store***

  "In this environment there is a particular need for storage technologies that are always available**. For example, customers should be able to view and add items to their shopping cart even if disks are failing, network routes are flapping, or data centers are being destroyed by tornados. Therefore, the service responsible for managing shopping carts requires that it can always write to and read from its data store, and that** its data needs to be available across multiple data centers**.**

- ***Facebook Photo Storage System (Haystack)***

  "In large scale systems, failures happen every day. Our users rely on their photos being available **and should not experience errors despite the inevitable server crashes and hard drive failures. It may happen that an entire datacenter loses power or a cross-country link is severed.** Haystack replicates each photo in geographically distinct locations**."**

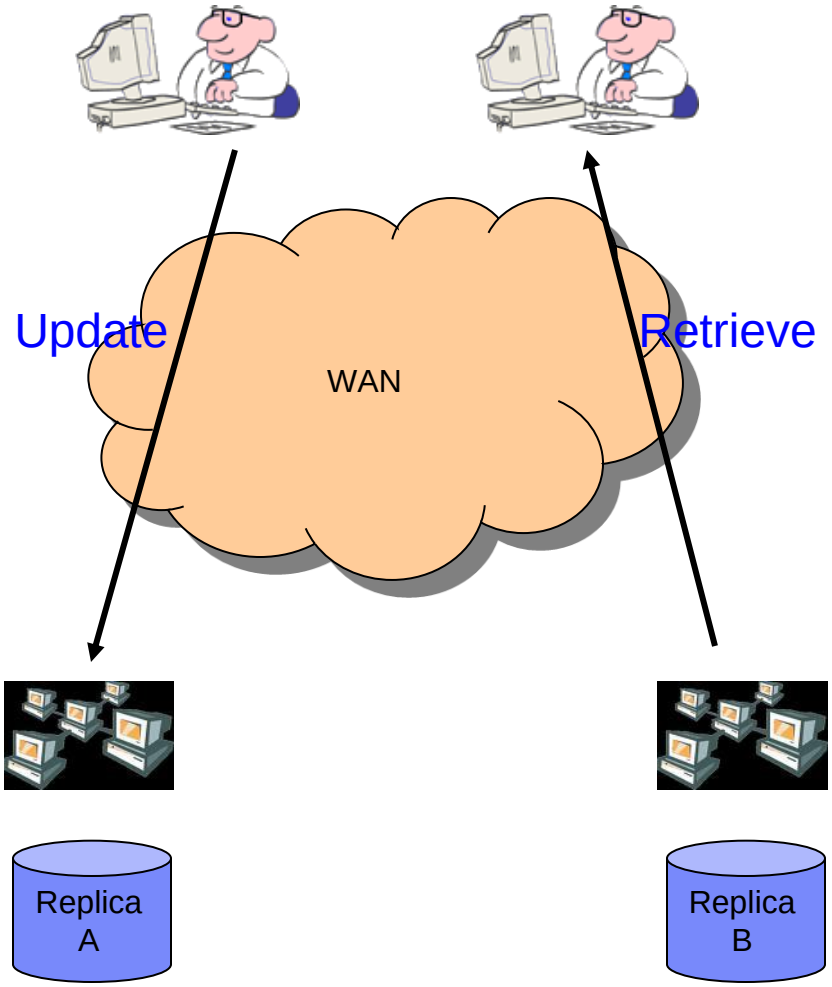
The CAP Theorem

- ***Lets concentrate on being:***
  - Always Available
  - Fault Tolerant (to many types of failure)
    - Requires replication across nodes/data centers

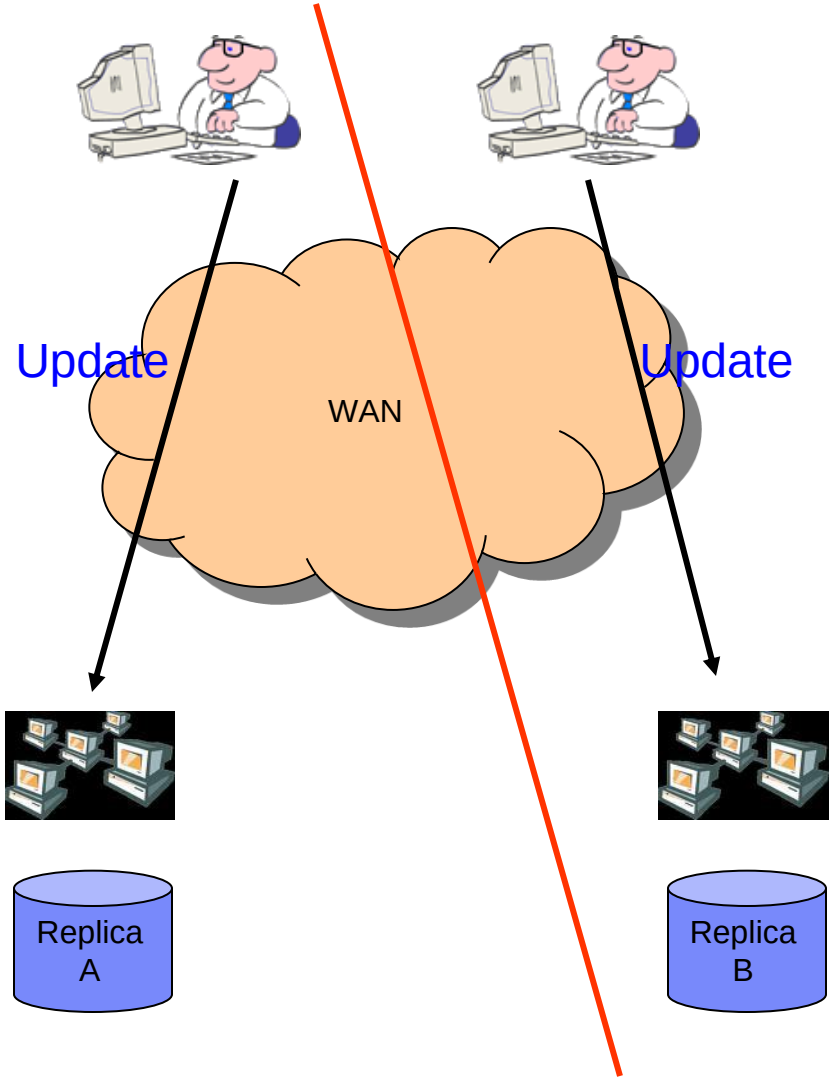
Is that asking too much?
Turns out that yes.

- ***The CAP Theorem (Eric Brewer):***
  - One can achieve at most two of the following:
    - Data Consistency
    - System Availability
    - Tolerance to network Partitions
  - Was first made as a conjecture At *PODC 2000 by* Eric Brewer
  - The Conjecture was formalized and confirmed by MIT researchers Seth Gilbert and Nancy Lynch in 2002

Consistency (Simplified)



Tolerance to Network Partitions / Availability



Problems with CAP

Daniel Abadi, Yale University

- ***Asymmetry of CAP properties***
  - C is a property of the system in general
  - A is a property of the system only if there is a partition
- ***Effectively, there are two choices here:***
  - Be Consistent (C):
    - Make sure you have a good transaction protocol. But in the presence of a network partition, the protocol would fail and result in un-availability
  - Be Available in the presence of partitions (PA):
    - This is where Cloud Storage typically lives. (Almost)
    - Always accept an update, no matter which replica got it.
    - But what with the consistency issue…

Adding Latency to the Game

Daniel Abadi, Yale University

- ***Consistency has its price even in the absence of network partitions***
- ***Synchronization between replicas has its overhead***
  - No way to get around without at least one round-trip message

CAP – Revised

Daniel Abadi, Yale University

- ***PALEC***
  - **In the case of a partition (P), does the system choose availability (A) or consistency (C)?**
  - **Else (E), does the system choose latency (L) or consistency (C)?**
- *PA/EL*
  - **Dynamo, SimpleDB, Cassandra, Riptano, CouchDB, Cloudant**
- *PC/EC*
  - **Traditional database systems**
- *PA/EC*
  - **GenieDB**
- *PC/EL*
  - **Existence is debatable**

Strong Vs. Weak Consistency

- *Strong consistency*
  - **after an update is committed, each subsequent access will return the updated value**
- *Weak consistency*
  - **the systems does not guarantee that subsequent accesses will return the updated value**
  - **a number of conditions might need to be met before the updated value is returned**
  - inconsistency window: **period between update and the point in time when every access is guaranteed to return the updated value**

Eventual Consistency

- ***A specific form of weak consistency***
- ***The Literature Definition:***
  - The storage system guarantees that if no new updates are made to the data eventually (after the inconsistency window closes) all accesses will return the last updated value.
- *In the absence of failures, the maximum size of the inconsistency window can be determined based on communication delays*
  - **communication delays**
  - **system load**
  - **number of replicas**
  - **…**
- ***Example:***
  - The most popular system that implements eventual consistency is DNS, the domain name system. Updates to a name are distributed according to a configured pattern and in combination with time controlled caches, eventually of client will see the update.

Models of Eventual Consistency

- ***Read your writes***
  - After updating a value, a process will always read the updated value, and will never see an older value
- ***Monotonic read consistency***
  - **if a process has seen a particular value, any subsequent access will never return any previous value**
- ***Monotonic write consistency***
  - **system guarantees to serialize the writes of one process**
- *Many more…*
- *Properties can be combined*
  - **E.g. monotonic reads plus read-your-own-writes**
- ***Applications should be written with the consistency model in mind.***

The Consistency-Availability Trade-Off

- ***Some Definitions:***
  - N – The number of replicas
  - W – The number of replicas that need to acknowledge the receipt of the update before the update completes
  - R - The number of replicas that are contacted when a data object is accessed through a read operation
- ***If W+R > N then the write set and read set are always overlapped and so consistency can be guaranteed.***
  - If however, not all replicas can be contacted, the system is not available.
  - Examples:
    - Systems that focus solely on fault-tolerance often use N=3 (with W=2 and R=2 configurations)
    - R=1,W=N. Optimized for read access.
    - W=1, R=N. Optimizes for write access
- ***If W+R <= N consistency cannot be guaranteed***

The Trade-Off Granularity

- ***Which of the following are natural to the eventual consistency model?***
  - Write an object as a whole
  - Append
  - Rename
  - ACL Setting
- ***Conclusion***
  - The Storage System can and should define different R, W settings for different operations.

References

- S. Gilbert and N. Lynch: **Brewer's Conjecture and the Feasibility of Consistent, Available and Partition-Tolerant Web Services.** *SIGACT News 33(2), pp. 51-59, 2002.*
- W. Vogels: **Eventually Consistent.** *ACM Queue 6(6), pp. 14-19, 2008.*
- *Daniel Abadi, Yale University*, ***Problems with CAP,*** http://dbmsmusings.blogspot.co.il/2010/04/problems-with-cap-and-yahoos-little.html