Cloud Applications

Geographically distributed applications: large-scale load balancing and eventually consistent data replication strategies

Reference:

- **Site Reliability Engineering - Chapters 19 and 23 by by Betsy Beyer, Chris Jones, Jennifer Petoff and Niall Richard Murphy**

  https://landing.google.com/sre/sre-book/toc/index.html

Global-scale applications

- Let us consider a cloud application available worldwide to hundreds of millions of users at the same time all around the world
- For instance, the Google Web Search engine, available worldwide and serving 5.4 billion search requests each day
- In order to handle such load, a VM cluster is required to load balance the requests among them and scale
- Regardless of the size of the cluster, however, this configuration might not be sufficient: the maximum number of requests is limited by the physical constraint of the network infrastructure that transmit the data from the clients to the VMs
- In order to mitigate this bottleneck the VMs of the cluster are scattered across different locations around the world, i.e. different datacenters
- The result is that the load is naturally divided across different locations, with different network infrastructures

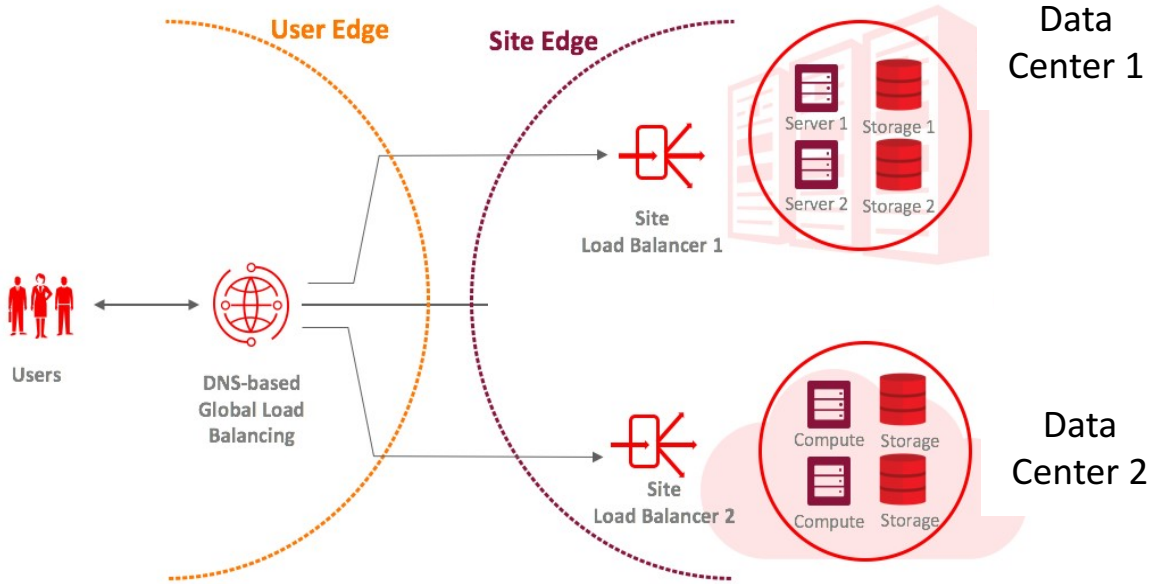
Load Balancing

- Load balancing is performed at two levels:
  - Global level: to dispatch (globally) the requests across different datacenters
  - Local level: to decide which VM serves a particular request within the datacenter
- *At the local level the load balancing policy usually aims at balancing the load for scalability*
- At the global level different more complex policies can be implemented depending on the application:
  - The request of a search service could be sent to the nearest available datacenter (measured in terms of round-trip time latency) in order to minimize the latency of the request
  - A video upload stream, instead, could be routed to the datacenter with the path that is less utilized, so to maximize the throughput at the expense of the latency to minimize the upload time

EXAMPLES

Global Load Balancing using DNS

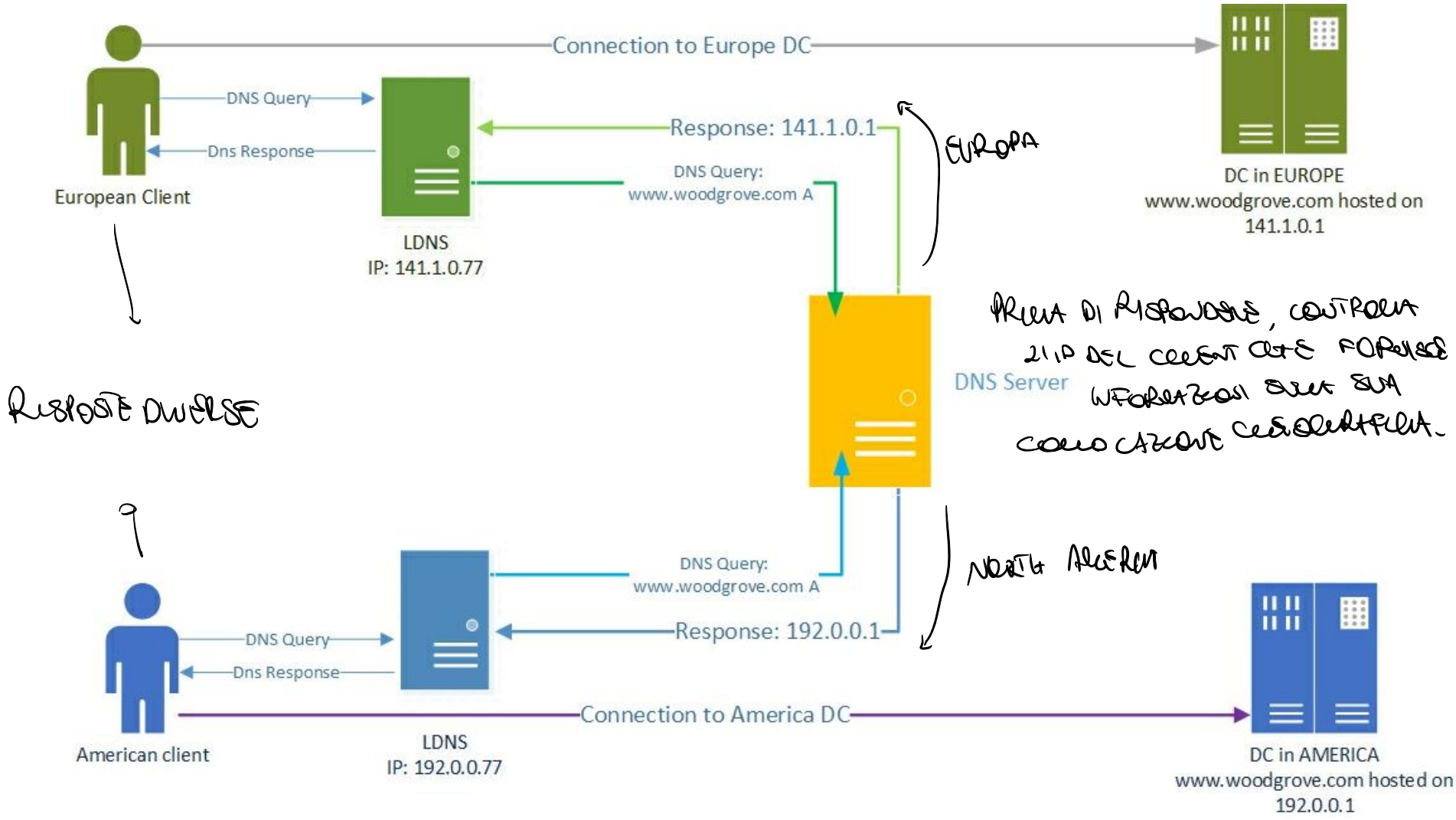
- Before the client can send its request, it has to lookup an IP address using DNS
- DNS load balancing exploit this phase to introduce a first layer for request dispatch
- The simplest solution is to configure the DNS records to return multiple A or AAAA records in the DNS reply and let the client pick an IP address arbitrarily



DNS Global Load Balancing - limitations

- Although simple, this solution has a main problem: it provides very little control over the client behavior
- *Records are usually selected by clients randomly as a client cannot determine which address is the closest*
- *Consequently, clients do not consider distance or other metrics for selection*
- To mitigate this issue a localized DNS system could be adopted to allow responses based on the geographical location of the client
- This method is the simplest and most effective, however, **it only offers a coarse-grained control**, as DNS records cannot be updated in real time (it takes hours to change a DNS record)
- A fine-grained control can be obtained by employing Virtual IPs

NESSUN CONTROLLO REALTIME


Connection to Europe DC
DNS Query
Dns Response
European Client
LDNS
IP: 141.1.0.77
Response: 141.1.0.1
DNS Query:
www.woodgrove.com A
DC in EUROPE
www.woodgrove.com hosted on
141.1.0.1
EUROPA
DNS Server
PRIMA DI RISPONDERE, CONTROLLA
L'IP DEL CLIENT CHE FORNISCE
INFORMAZIONI SULLA SUA
COLLOCAZIONE GEOGRAFICA.
RISPOSTE DIVERSE
DNS Query:
www.woodgrove.com A
Response: 192.0.0.1
NORTH AMERICA
DNS Query
Dns Response
American client
LDNS
IP: 192.0.0.77
Connection to America DC
DC in AMERICA
www.woodgrove.com hosted on
192.0.0.1

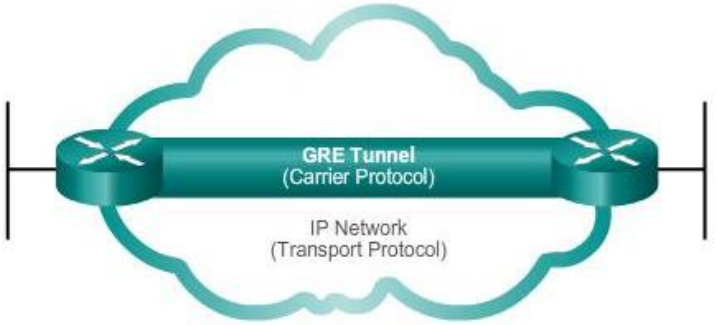
Virtual IP Addresses

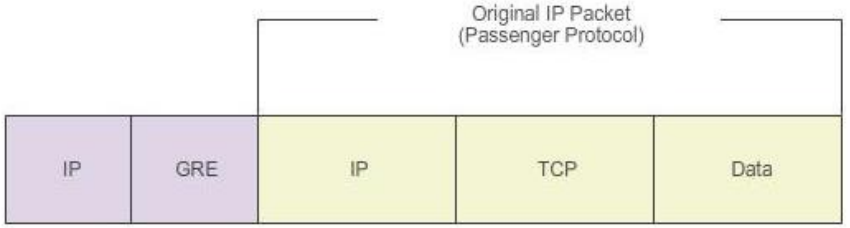
IP NORMALE CHE NON È ASSEGNATO AD UN HOST, BENSÌ AD UNA APPLICAZIONE O AD UNA CERTA AREA.

- A Virtual IP address (VIP) is an IP address that is not assigned to any host or any particular network interface and it is shared among different devices
- From the point of view of the user it remains a regular IP address, hiding all the implementation details
- A set of VIPs could be assigned to an application (e.g. the web search engine) and each VIP (or a subset) could be assigned to a region, in each region (e.g. wester Europe) multiple datacenters could be available
- The regional DNS service is configured with the VIPs assigned to that region for a specific service
- For each region one (or more) global load balancer are deployed. A global load balancer could be installed within the ISP networks or in certain exchange points (where different ISPs interconnect)
- Each global load balancer has assigned one VIP from the set of the VIPs assigned to the region

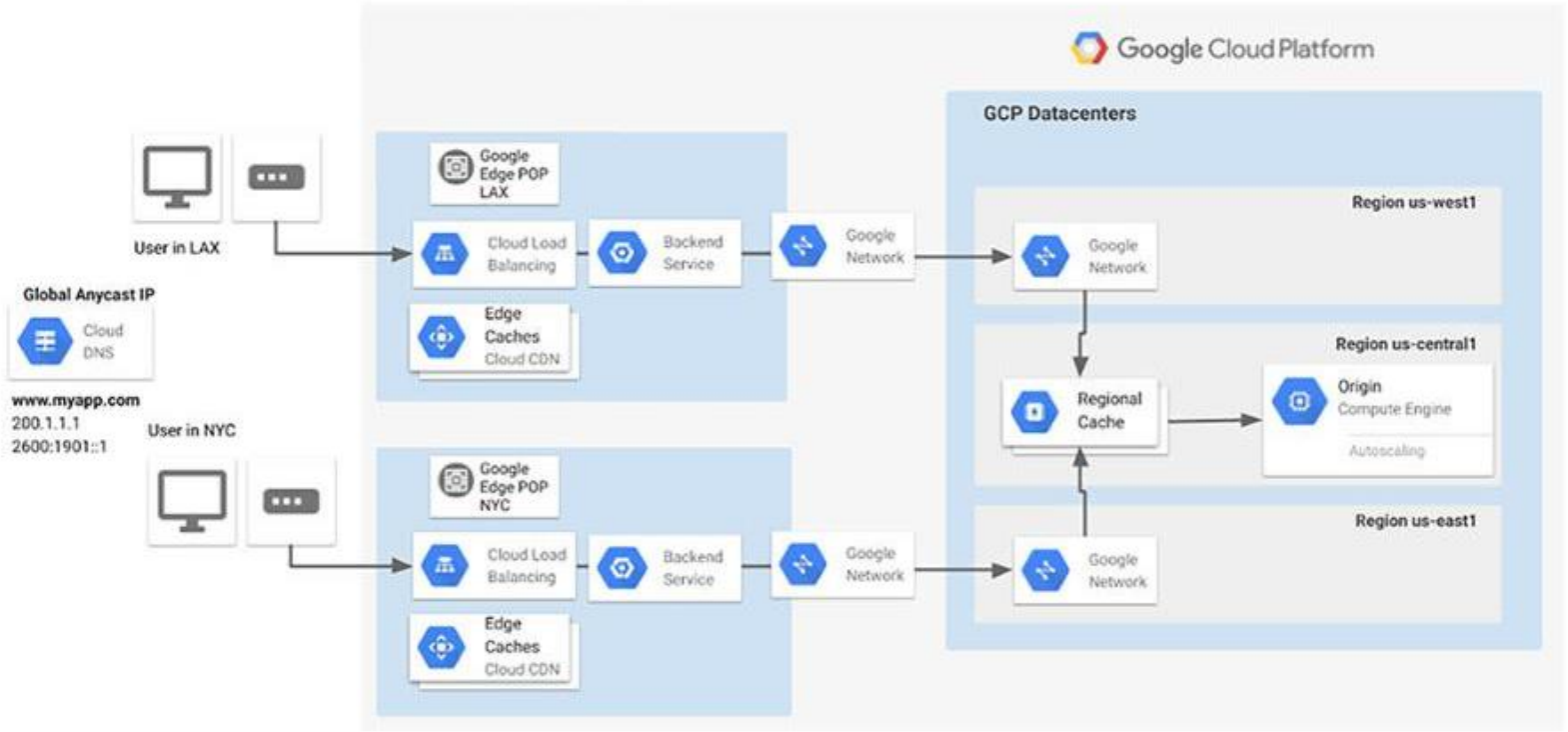
Packet Encapsulation

- The global load balancer is responsible for receiving the requests and dispatching them to the more convenient data center in that moment (the selection could change every minute)
- This is performed by means of packet encapsulation: the packet is encapsulated into another packet and sent to the selected datacenter
- At the destination datacenter the packet is decapsulated and processed like it were received directly by the datacenter
- These operations are usually performed by the datacenter local load-balancer
- Generic Routing Encapsulation (GRE) is usually adopted to establish a tunnel between the global load balancer and the datacenter
- GRE allows to encapsulate an IP packet inside another IP packet
- The outer IP packet contain the destination IP address of the load-balancer at the destination datacenter





Load Balancing using Virtual IP Addresses



Geographically distributed systems

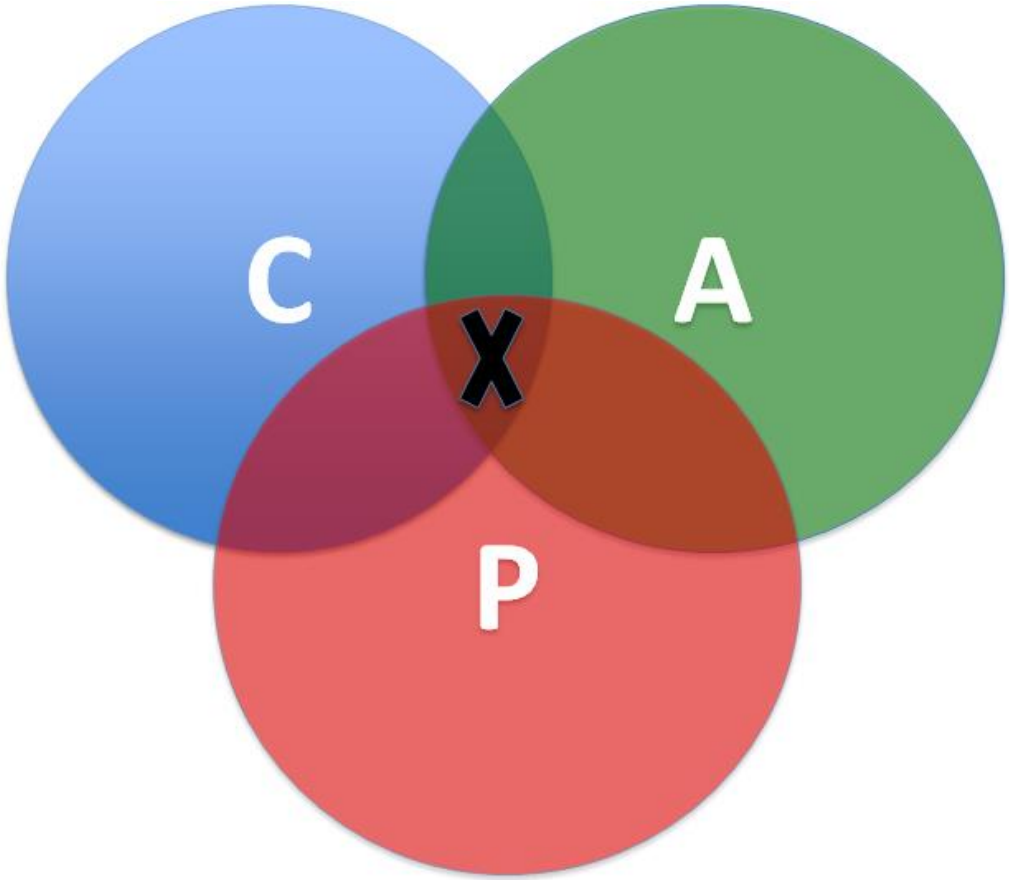
- Globally distributed systems need to have a consistent view of the system state and of the data across multiple sites
- Data replication and synchronization is a challenge across distant sites interconnected via wide area network links due to
  - Limited bandwidth, which increases significantly the time required for data synchronization operation
  - Lower communication reliability, which can result in frequent network partitions
- For those reasons, traditional data replication strategies are not well suited for systems that work across different sites as they usually assume a reliable network with low latency for timed data exchange
- ***Network partitions in particular are not considered in such data replication strategies***: a node can disconnect for a certain time, however, during disconnection it cannot offer its service; when it comes back online it synchronizes with the other nodes assuming that it was offline (no data was generated)
- ***In a geographically distributed system you must tolerate network partition***: a certain region (or a datacenter) could disconnect from the others for a certain period of time, during which **they must continue to offer the service**
- This is because network partitions in such systems are inevitable, e.g. cables get cut, packets get lost or delayed due to congestion, networking components become misconfigured, etc…

ACID Systems

- In traditional systems (e.g. an application with a relational database) data is stored according to the ACID semantic: Atomicity, Consistency, Isolation and Durability
- Such systems focuses on Consistency and (secondarily) availability
- Availability (to tolerate failing nodes) can be ensured through replication (a replicated master/slave database), which, however, is performed in a rigorous manner in order to ensure strict consistency. This approach is the one adopted by traditional databases
- All the data replications techniques for such systems assumes that network partitions are rare, e.g. when all the replicas are on the same LAN (the same datacenter). When a partition occurs the partitioned replicas are considered failed and stop providing the service
- In a global scale in which partitions can be frequent and each node serves thousands of clients, partitioned replicas must go on and provide the service
- For this reason, a different approach is needed to handle network partitions!

CAP Theorem

- A distributed system can have the following properties:
- **C**onsistency:
  - All nodes should see the same data at the same time
- **A**vailability:
  - *Node failures* do not prevent survivors from continuing to operate
- **P**artition-tolerance:
  - The system continues to operate despite *network partitions*
- *The CAP theorem: a distributed system can satisfy any two of these guarantees at the same time but not all three*
- *Note that network partitions are different from node failure, one node of the system broke vs the distributed system is partitioned!*

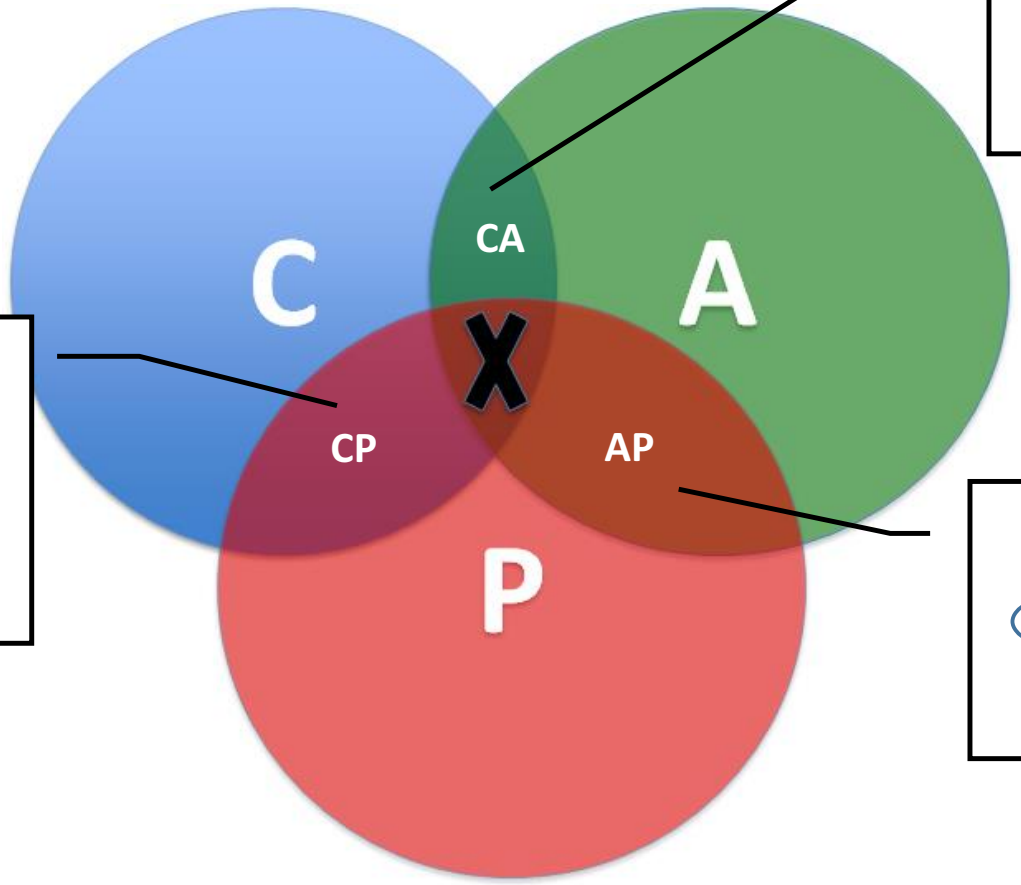


Design trade-off

- A proper understanding of CAP theorem is essential to making decisions about the design of distributed systems as misunderstanding can lead to erroneous or inappropriate design choices
- *CAP theorem describes the trade-offs involved in distributed systems*
- *Since a CAP system is unfeasible, only three types of distributed systems are possible (in theory):*
  - *CP: Consistency - Partition Tolerance*
  - *AP: Availability - Partition Tolerance*
  - *CA: Consistency - Availability*
- *CAP prohibits only a tiny part of the design space: it states that **perfect** availability and consistency in the presence of partitions are not feasible, it does not mean that availability or consistency are not considered at all*
- *Consistency and Availability is not "binary" decision: AP systems relax consistency in favor of availability – but are not inconsistent, while CP systems sacrifice availability for consistency- but are not unavailable*
- *AP and CP systems offer a degree of consistency, and availability, as well as partition tolerance*

Classification

Existing distributed systems can fit in this classification. Their classification sometimes is not unanimous



ACID Systems
Relational DataBase Management Systems
RDBMS

Election based systems (e.g. Zookeeper), MongoDB

Best Effort Availability

DNS, Web Cache,
BASE Systems

Best Effort Consistency

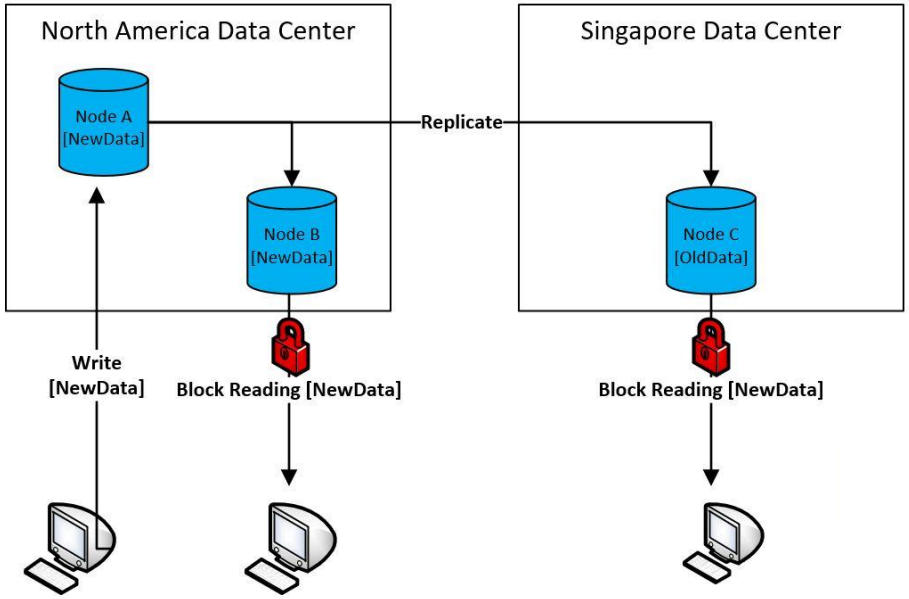
BASE semantic

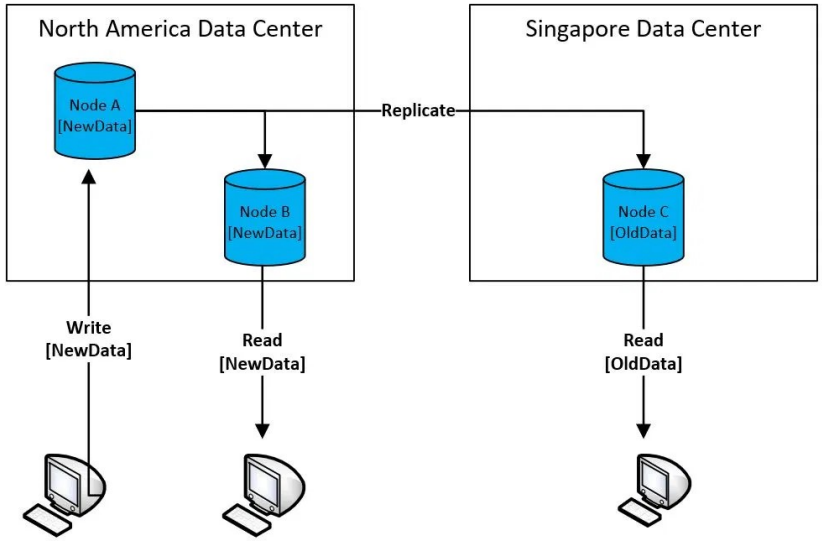
- A different semantic has been defined recently for systems that handle large volumes of data for which the ACID semantic would be unfeasible or too expensive: ***Basically Available, Soft state and Eventual consistency – BASE***
- At the core of the BASE semantic there is the concept of eventual consistency: the system status and its data can be inconsistent for a certain period of time, however, it is guaranteed that consistency will be reached eventually after a transition period
- This is opposed to the Strong Consistency concept, in which is strictly guaranteed that after an update, any subsequent access will return the same updated value.
- This approach, of course, can work for all the applications that can tolerate not correct data for their functionalities during the transition periods (e.g. a social media website), but it is unfit for all the applications that require consistency all the time (e.g. a bank system)

Query/Update

- The main difference among the two semantics is in how **updates** are handled:
  - In ACID systems data updates are propagated immediately in a synchronous manner. The result is that the user waits for the completion of the propagation of the new data (at least this happens in most of the replication strategies)
  - In BASE systems, instead, data is replicated asynchronously in background: the request from the user is handled immediately and the value is updated locally. The user does not have to wait long as a response is generated immediately. The update is propagated in a deferred manner using a specific protocol, named ***anti-entropy protocol***
- Query operations, instead, are handled simply by providing an answer based on the local status

ACID vs BASE


North America Data Center
Singapore Data Center
Node A
[NewData]
Replicate
Node B
[NewData]
Node C
[OldData]
Write
[NewData]
Block Reading [NewData]
Block Reading [NewData]


North America Data Center
Singapore Data Center
Node A
[NewData]
Replicate
Node B
[NewData]
Node C
[OldData]
Write
[NewData]
Read
[NewData]
Read
[OldData]

Inconsistency

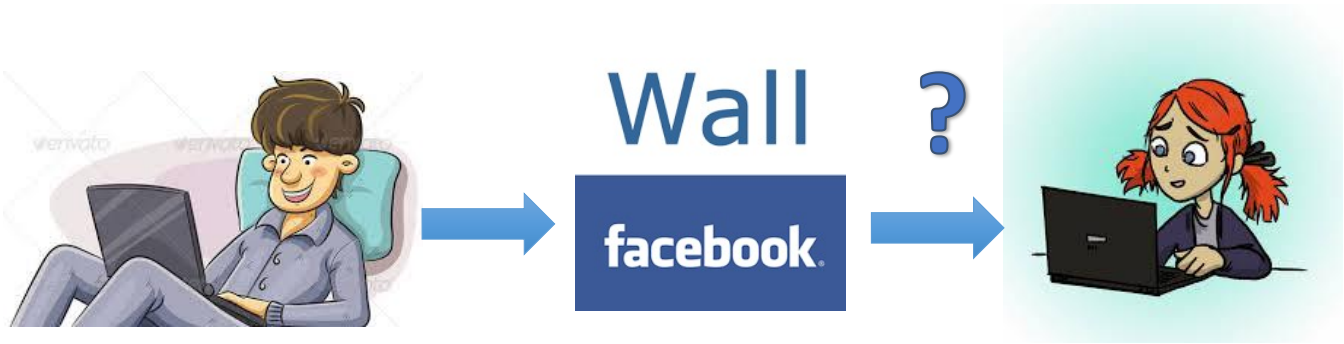
- Due to this deferred propagation of the information, inconsistencies might arise
- Query operations can return inconsistent data:
  - This can occur when a specific update has not reached the site that manages the query request
- This was also the case of the Gossip architecture, which could return outdated data during transient
- In eventually consistent systems data conflicts can arise, as two different sites can accept two conflicting values:
  - This can happen when two concurrent update request are received and stored in two different locations
- When a site receives an update that conflicts with the local copy it must run a conflict resolution mechanism
- A mechanism to resolve conflicts with a certain policy must be defined, e.g. the update with the last timestamp is accepted while the other is discarded

Advantages/Disadvantages

- This mechanism allows to ***handle network partitions***
  - If a network partition occurs in the global network (or simply the updates are delayed due to network congestion) a certain region or a certain datacenter can continue to operate
  - When the partition is resolved the anti-entropy protocol will distribute the data eventually reaching a consistency point
- ***Response delay*** is minimal
  - A client does not have to wait for the system to synchronize, which is a significant overhead especially for global scale systems
- This approach is not suitable for systems that do not tolerate conflicts

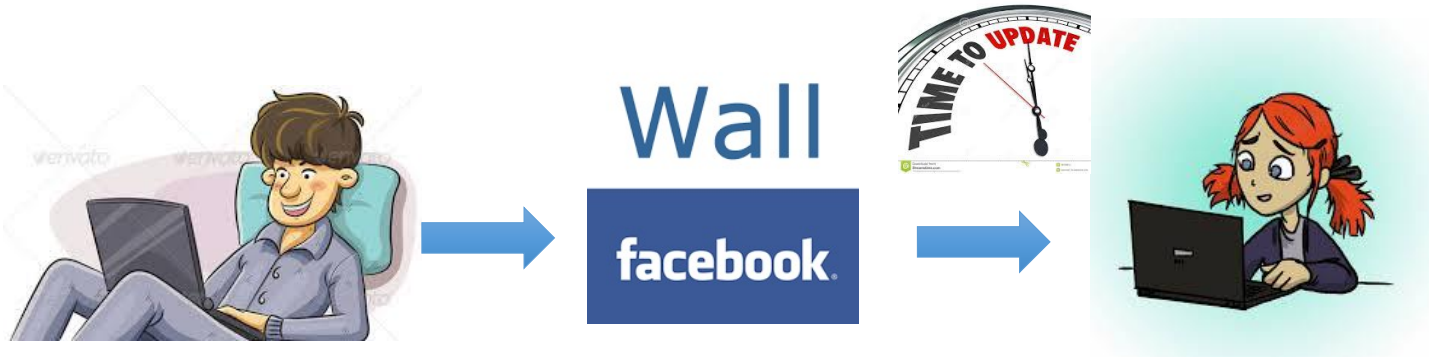
Eventual Consistency - A Facebook Example

- Bob finds an interesting story and shares with Alice by posting on her Facebook wall
- Bob asks Alice to check it out
- Alice logs in her account, checks her Facebook wall but finds:
  - **Nothing is there!**



Eventual Consistency - A Facebook Example

- Bob tells Alice to wait a bit and check out later
- Alice waits for a minute or so and checks back:
  - **She finds the story Bob shared with her!**



Eventual Consistency - A Facebook Example

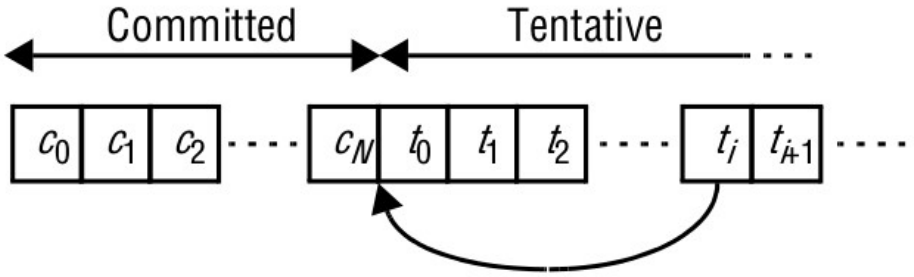
- Reason: Facebook uses an **eventual consistent model**
- Why Facebook chooses eventual consistent model over the strong consistent one?
  - Facebook has more than 1 billion active users
  - It is non-trivial to efficiently and reliably store the huge amount of data generated at any given time
  - Eventual consistent model offers the option to **reduce the load and improve availability**

Bayou architecture

- The Bayou architecture is an eventually consistent architecture that provides data replication for high availability ***with weak guarantees on sequential consistency***
- Each replica manager exchange updates on the data using an anti-entropy protocol that provides updates in group
- The architecture includes also a domain-specific conflict resolution policy that is used to resolve conflicts
- It ensures that through the continuous exchange of messages that the data on all the replica managers is the same

Bayou operations

- When an update is received, it is initially applied but marked as ***tentative***
- *While an update is tentative, the system may undo and reapply it as necessary to reach a consistent state*
- *Instead once an update is committed it remains applied*
- At any time the status of the system derives from the sequence of committed updates followed by the sequence of tentative updates
- Committed updates are ordered in their order of commitment
- Tentative transactions are ordered based on the local reception timestamp



Tentative update t_i becomes the next committed update and is inserted after the last committed update c_N.

Primary replica manager

- Bayou adopts a primary commit scheme i.e. one replica manager is selected as primary and takes responsibility for committing the updates
- The list of the updates that are committed is exchanged via the anti entropy protocol
- Other than deciding when an update is committed the primary replica manager operates as the others
- The commit protocol can be any policy that establishes the order in which updates are applied and it can be defined specifically to suit the requirements of the application
- The temporary unavailability of the primary (e.g. due to a disconnection of a replica server) does not prevent requests to be handled:
  - Updates are accepted as tentative in the order they are received
  - Queries are answered using the current state of the system
- When the disconnected replica manager connects again it commits the updates

Conflict detection and resolution

- When an update (tentative or committed) is applied a conflict might arise
- Bayou adopts two mechanism to detect and resolve conflicts that can be defined by application developers:
  - A *dependency check: a query and the expected result* provided by the developer, to check if a set of pre-conditions are met for the execution of the update. Before applying the update, the replica manager executes the query. If the results differ with the ones provided by the developer a conflict is arisen
  - A *merge procedure*, which is a procedure that is run when a conflict is detected. The merge procedure are provided by the developers
- Both dependency check and merge procedures are deterministic so that every replica manager resolves the same conflict in the same manner as the others
- As a result, the execution history and the conflict resolution are the same on all the replica manager

Performance measurement

- Eventually consistent systems are widely adopted today, even though they don't provide safety, it is the proof that such systems can work in practice and has multiple advantages in terms of latency and availability
- A common metric to measure eventual consistency is time:
  - The window of inconsistency
  - The time required to have an information visible on the system
- Both of them are dependent on the anti-entropy protocol and in particular the anti-entropy rate, the frequency to which the anti-entropy protocol sends the update
- Given a certain anti-entropy rate the expected time to consistency can be calculated in order to obtain a certain Probabilistically Bounded Staleness (PBS)
- Many existing system allows to set some system parameters to obtain a certain PBS