Mobile Cloud Computing

EINSTEIN CENTER Digital Future

Technische Universität Berlin

Fog Computing

Bermbach | Part 3: Data Management

Agenda

Technische Universität Berlin

Lectures

- **From Cloud to Fog Computing**
- **Data Distribution**
- **Data Management**
- **Platforms & Applications**
- **Testing & Benchmarking**
- **LEO Edge Computing**

Assignments

- Prototyping Assignment
- Reading Assignment

Wrap-up

- **Q&A**
- Final Test

Mobile Cloud Computing

EINSTEIN CENTER Digital Future

Data Management

In Fog Computing, the number of nodes and their geographic distribution increases. What does this mean for data management?

Technische Universität Berlin

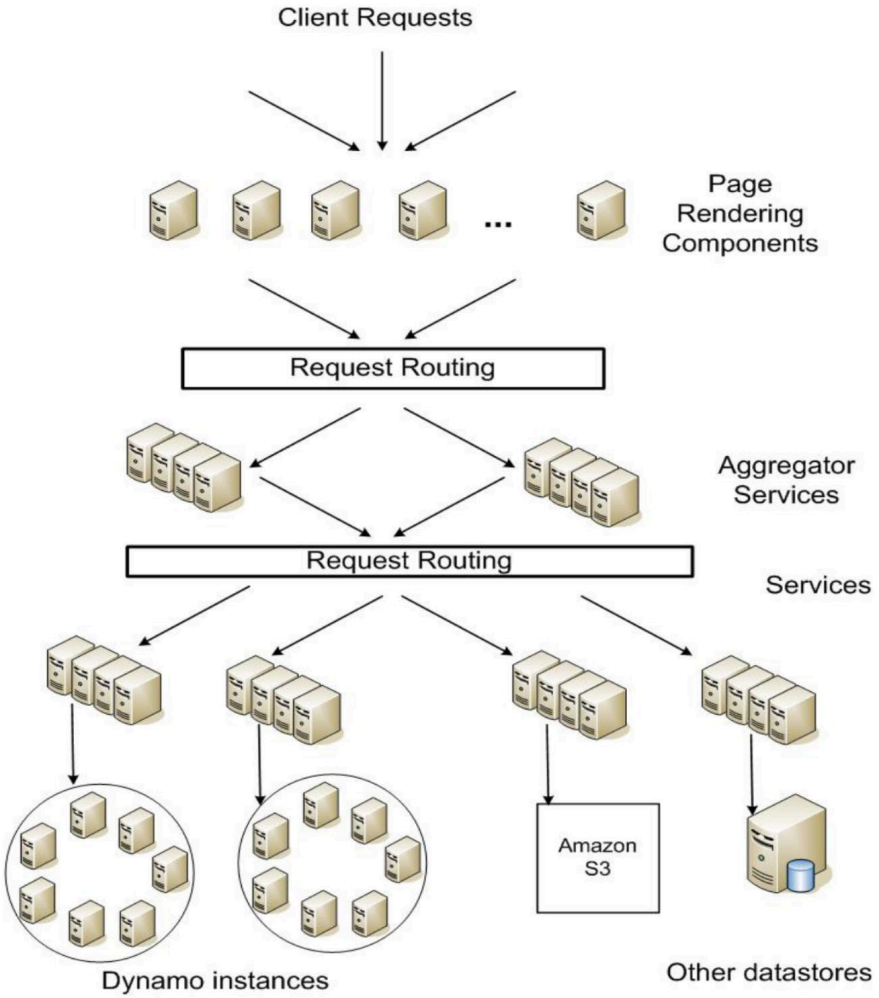
Replication

OVERVIEW & RECAP

Mobile Cloud Computing

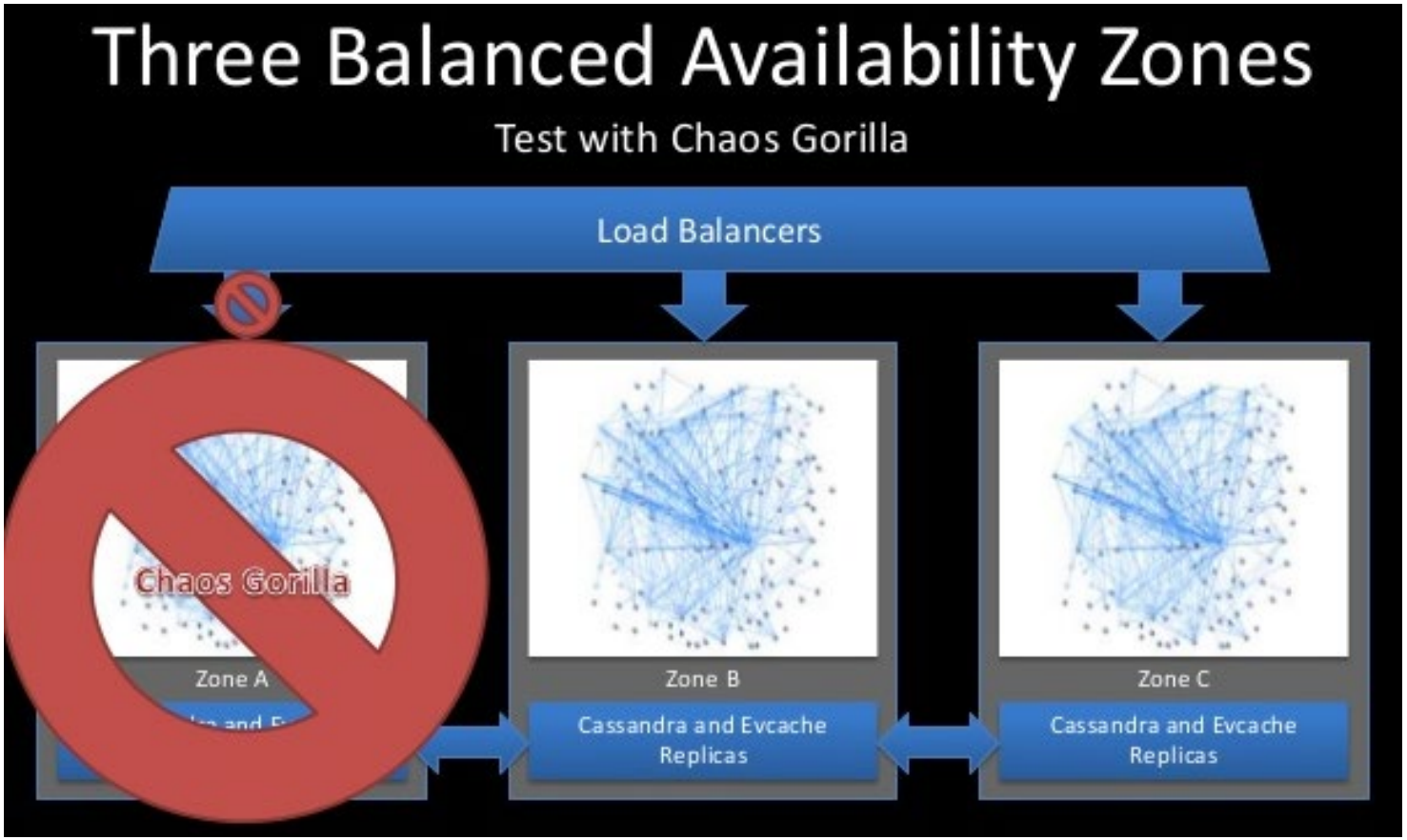
EINSTEIN CENTER Digital Future

Replication: Amazon‘s Dynamo



Source: amazon.com

Replication: Tolerating Outage of an Entire Availability Zone at Netflix



Source: Adrian Cockcroft, Netflix

What is Replication?

Replication is a common strategy in data management and in distributed systems.

The main idea is to use and to maintain multiple copies of an entity (data, process, file, etc) – called replicas – on multiple servers for better availability and performance.

BUT there is a price for replication: Keeping replicas consistent in face of updates can be costly (and may even negatively impact performance).

Why Replication?

Technische Universität Berlin

System availability / Fault-tolerance

- Failure resilience is a critical aspect in any enterprise system. If a server fails, the data it contains becomes unavailable, leading to potentially significant consequences for the enterprise.
- By keeping several copies of the server single failures should not affect overall availability; redundancy allows to switch-over in case of failures.

- Availability is also related to security. Replicas can provide protection against corrupted data (➔voting)

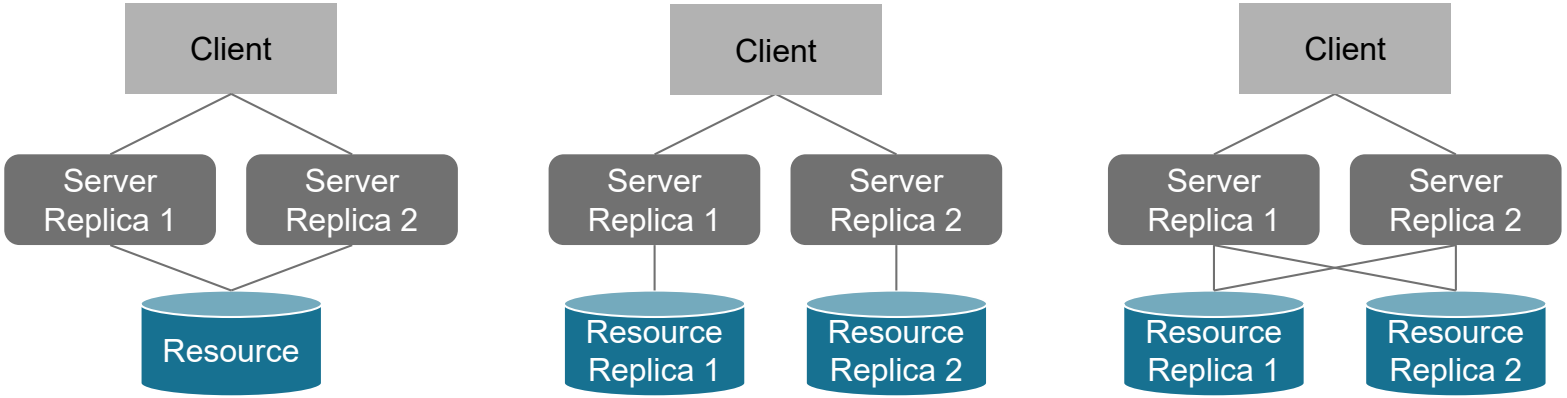
Performance / Scalability

- Large workloads can be spread and balanced across distributed replicas.
- Local access is fast, whereas remote access is slow: Place copies of data and processes in client proximity

Mobile Cloud Computing

EINSTEIN CENTER Digital Future

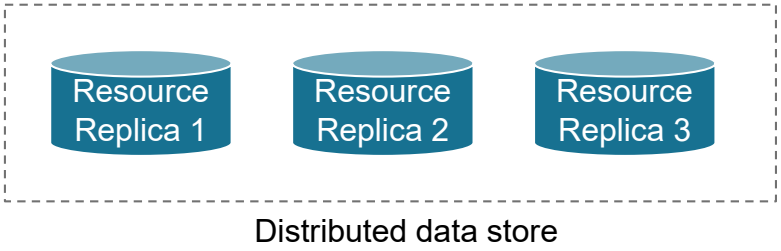
Replicating Servers and Resources



- Replicating servers on a common resource (A) may help availability if there is a replica cache coherence mechanism.
- To get improvement in availability the resources must be replicated, too (B). This is what is usually meant by replication.
- Replicated servers and resource replicas are not necessarily tightly coupled (C).

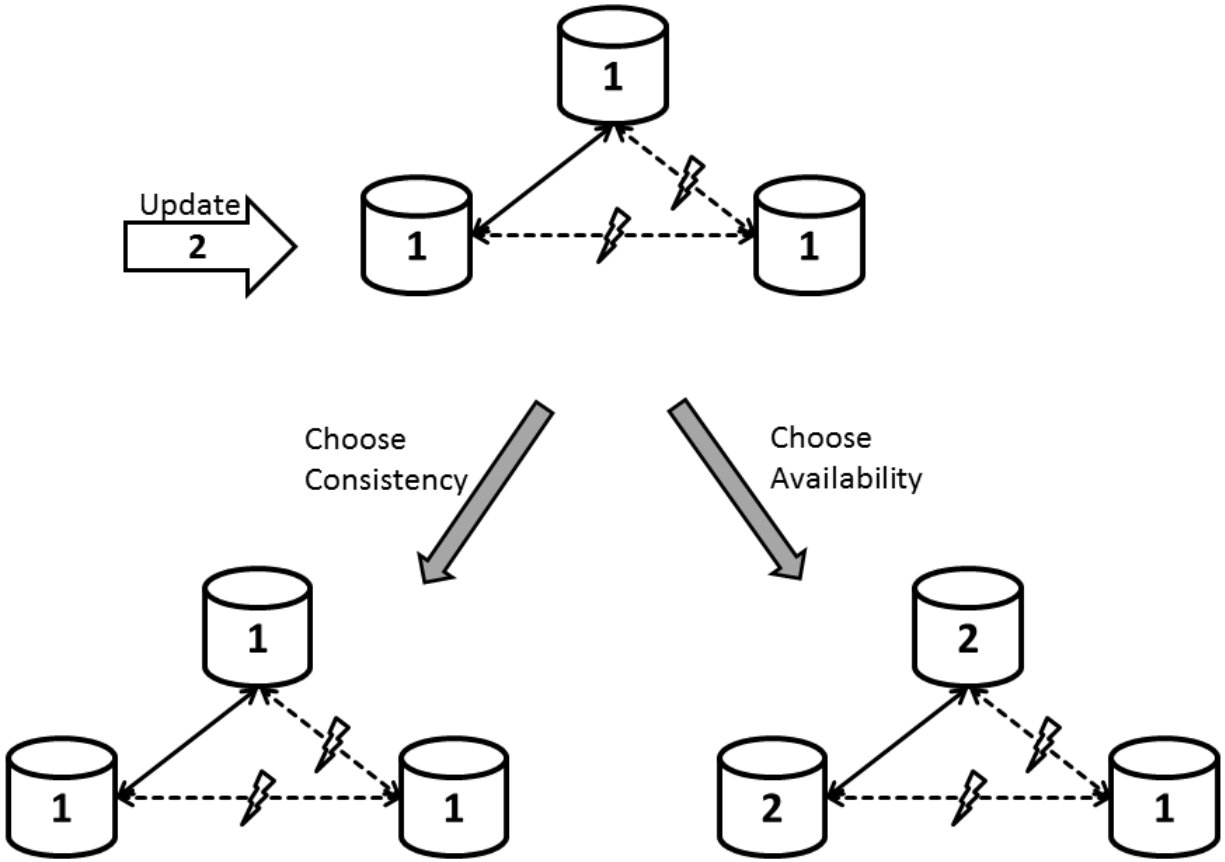
DATA CONSISTENCY

Replica Consistency

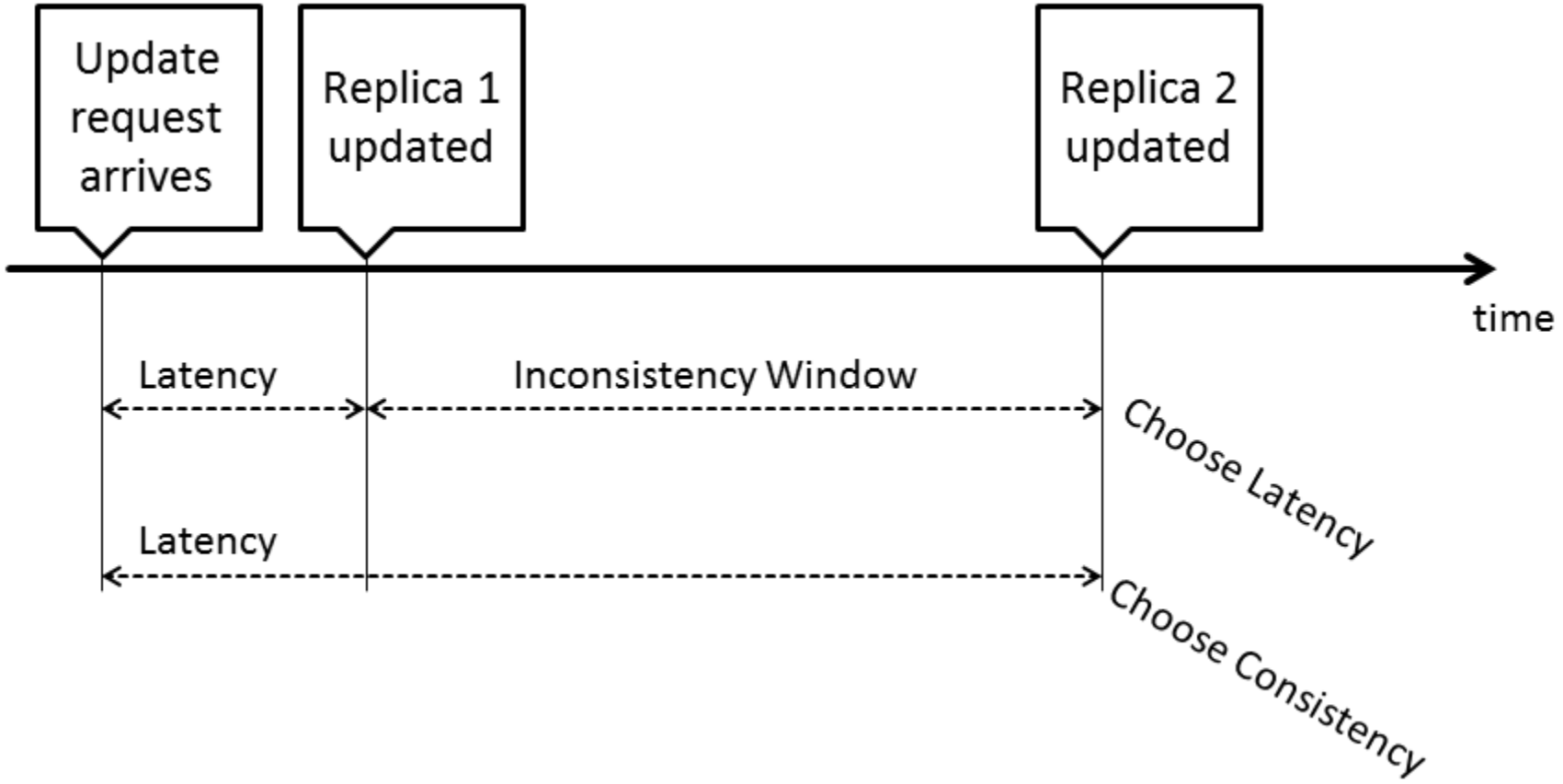


- Entities of a single logical data store are physically distributed and replicated across different machines
- Normally one would like that a *read* [to any replica] returns the result of the *latest write to the [logical] data store*
- Consistency of all replicas is needed, but this is expensive. Different consistency models exist, each of which relaxes specific requirements to a different extent (will be covered later)

Meet CAP…


Update
2
1
1
1
Choose
Consistency
Choose
Availability
1
1
1
2
2
1

…and PACELC

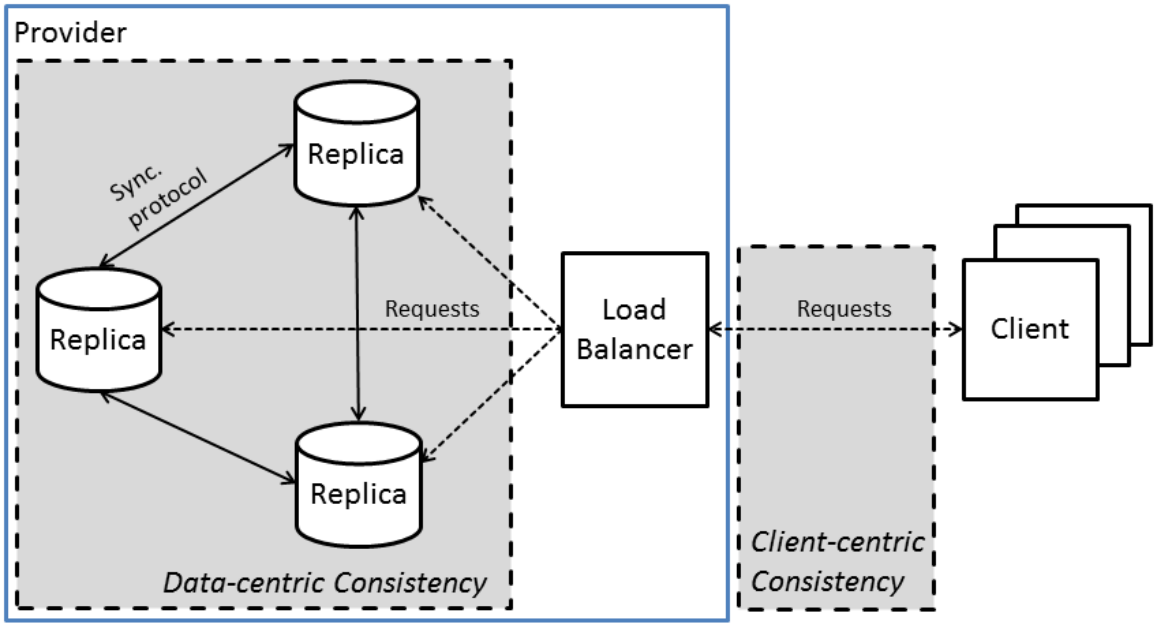

Update request arrives
Replica 1 updated
Replica 2 updated
time
Latency
Inconsistency Window
Choose Latency
Latency
Choose Consistency

Characterizing consistency

There are two dimensions of consistency…

- Staleness: “How much is a given replica lagging behind?“
- Ordering: “How much does the operation serialization order deviate among replicas?“

…and two groups of models:



Data-centric consistency models (ordering)

take the perspective of a database provider.

Technische Universität Berlin

Sequential Consistency: All replicas execute all updates in the same order.

Causal Consistency: All replicas execute causally-related operations in the same order; concurrent requests are executed in arbitrary order.

Example:

w(7) and w(8) are concurrent and not causally-related; w(9) and w(10) are causally-related.

Eventual Consistency: In the absence of updates and failures, all replicas converge towards the same state.

There are even more…

Mobile Cloud Computing

EINSTEIN CENTER Digital Future

Client-centric consistency models (ordering)

take the perspective of a single database client.

Technische Universität Berlin

Monotonic Reads: A read will never return older values than previously returned to the same client.

Read Your Writes: A read will never return older values than previously written by the same client.

Write Follows Reads: A client that reads version X and then updates the same data item, will only update replicas that have at least version X.

Monotonic Writes: Two updates of the same client will always be serialized corresponding to the chronological order of their submission.

"Systems that do not guarantee this level of consistency are notoriously hard to program." Werner Vogels, Eventually Consistent, ACM Queue, 2008.

Mobile Cloud Computing

EINSTEIN CENTER Digital Future

Examples of effects on an email application

No monotonic reads:

"When reloading your email inbox, previously seen emails will suddenly disappear."

No read your writes:

"When you open the sent folder, the email you just sent may not be there."

No monotonic writes:

"After creating an email draft, you remove person X from the list of addressees. When you send the email you realize that the email was sent to the old list of recipients – including person X."

…

To wrap up…

While applications benefit from high availability and low latency through replication, they are also (potentially severely) affected by inconsistencies.

Depending on the concrete applications, inconsistencies of a different kind may or may not be acceptable (staleness often is, ordering is more problematic).

Applications need to be able to deal with inconsistencies and conflicts.

Reconciliation in the storage layer is rarely possible without loss of information. If inconsistencies are unavoidable, there resolution is, thus, often shifted to the application layer.

REPLICATION STRATEGIES

How to Replicate Data?

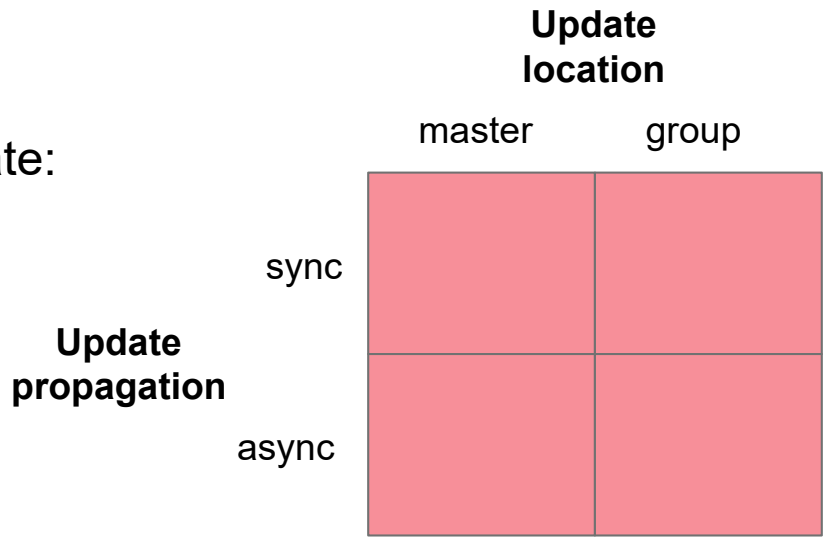
There are two basic parameters to select when designing a **replication strategy**: when and where.

Depending on when updates are propagated:

- **Synchronous (eager)**
- **Asynchronous (lazy)**

Depending on where updates can originate:

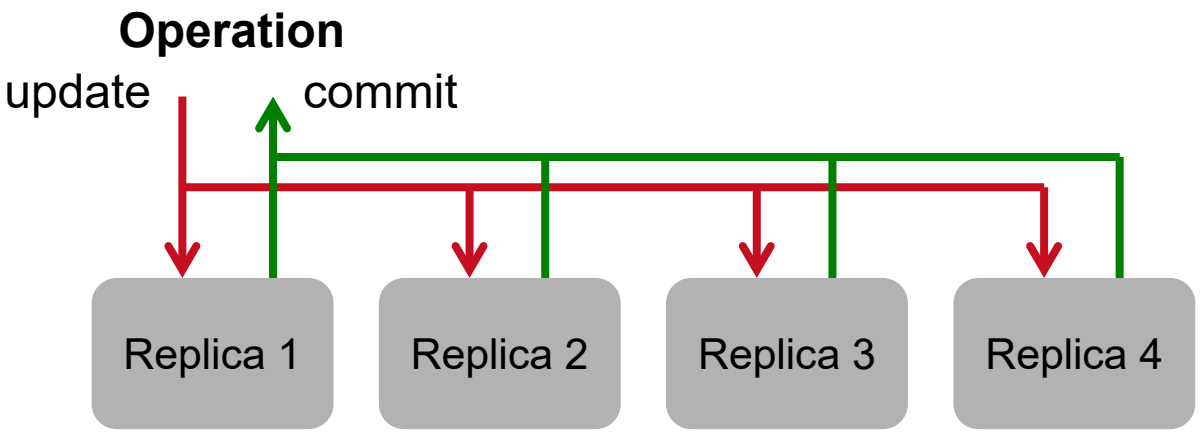
- **Primary copy (master)**
- **Update everywhere (group)**



Synchronous Replication

Synchronous replication propagates any changes to the data immediately (before the commit) to all existing copies.

Moreover, the changes are propagated within the scope of the operation making the changes. The ACID properties can apply to all replica updates.



Synchronous Replication

With synchronous replication, data copies are consistent (that is, identical) at all times and at all sites.

If one site wants to update the data, it first consults with everybody else and only if an agreement among all sites is reached the data is updated.

However: The system is unavailable for updates if only a single replica cannot be reached.

Asynchronous Replication

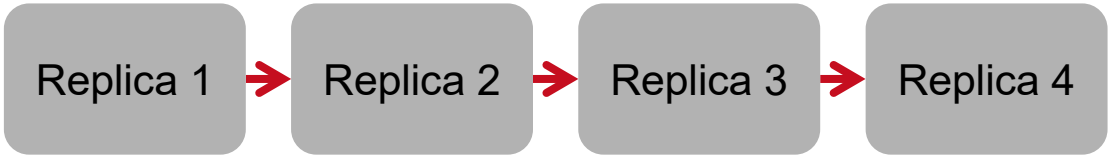
Asynchronous replication first executes the updating operation on the local copy. The operation commits before propagating the changes.

Next, the changes are propagated to all other copies. While the propagation takes place, the copies are inconsistent (that is, they have different values).

Asynchronous Replication

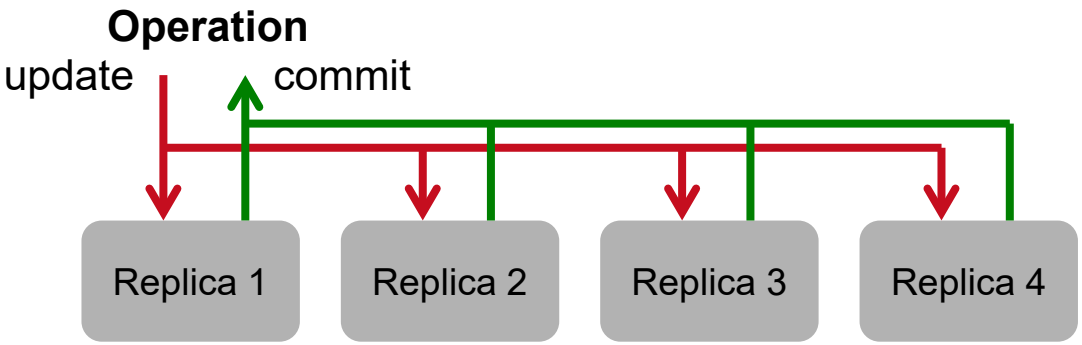
Initially, we assume data to be consistent at all sites.
If one site wants to update the data it does so locally and continues processing.
The data at all sites is no longer consistent.

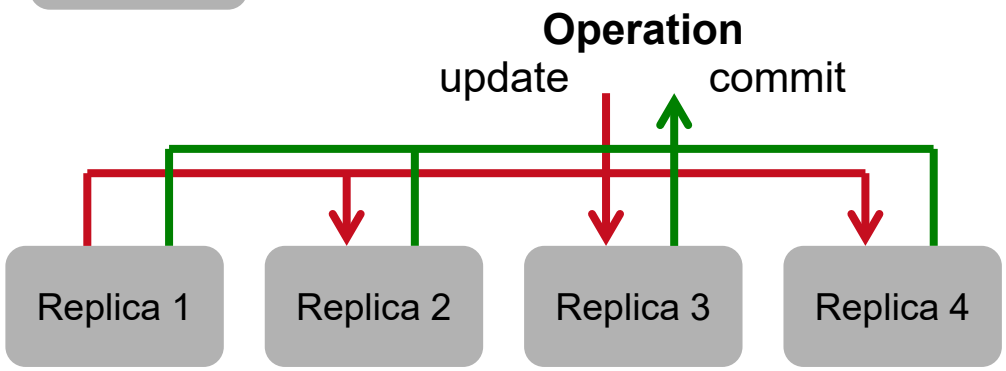
The update is eventually propagated to all sites (using push or pull models) and assuming that no conflicts arise the data eventually becomes consistent again.



Update Everywhere

With an **update everywhere** approach, changes can be initiated at any of the copies. That is, any of the sites which owns a copy can update the value of the data item.





Primary Copy

With a primary copy approach, there is only one copy where updates can originate (the master), all others (secondary copies) are updated reflecting the changes to the master.

All secondary copies, consequently, are read-only copies.

Forms of Replication

Technische Universität Berlin

Synchronous

Advantages:
- No inconsistencies (identical copies)
- Reading the local copy yields the most up-to-date value
- Changes are atomic

Disadvantages: An operation has to update all sites (longer execution time, worse response time, poor availability)

Asynchronous

Advantages: An operation is always local (good response time, high availability)

Disadvantages:
- Data inconsistencies
- A local read does not always return the most up-to-date value
- Changes to all copies are not guaranteed
- Replication is not transparent

Update Everywhere

Advantages:
- Any site can run an operation
- Load is evenly distributed

Disadvantages:
- Copies must be synchronized
- Concurrent updates will cause conflicts

Primary Copy

Advantages:
- No inter-site synchronization is necessary (it takes place at the primary copy)
- There is always one site which has all the updates

Disadvantages:
- The load at the primary copy can be quite large
- Reading the local copy may not yield the most up-to-date value
- There is a single point of failure (the master)

Mobile Cloud Computing

EINSTEIN CENTER Digital Future

The Previous Ideas can be Combined into Four Different Replication Strategies

	Primary copy	Update everywhere
Synchronous (eager)	**Synchronous Primary copy**	**Synchronous Update everywhere**
Asynchronous (lazy)	**Asynchronous Primary copy**	**Asynchronous Update everywhere**

...each with Advantages and Disadvantages

Technische Universität Berlin

Synchronous Primary copy Advantages: Updates do not need to be coordinated No inconsistencies Disadvantages: Long response time Only useful with few updates Local copies are read-only Low availability	**Synchronous Update everywhere** Advantages: No inconsistencies Elegant (symmetric solution) Disadvantages: Long response times Updates need to be coordinated Low availability
Asynchronous Primary copy Advantages: No coordination necessary Short response times Disadvantages: Local copies are not up-to-date Inconsistencies Low write availability	**Asynchronous Update everywhere** Advantages: No centralized coordination Shortest response times High availability Disadvantages: Inconsistencies and conflicts Updates can be lost (reconciliation)

Mobile Cloud Computing

EINSTEIN CENTER Digital Future

Replication – Ideal

	Primary copy	Update everywhere
Synchronous (eager)	**Globally correct Remote writes**	**Globally correct Local writes**
Asynchronous (lazy)	**Inconsistent reads**	**Inconsistent reads Reconciliation**

Replication – Practical

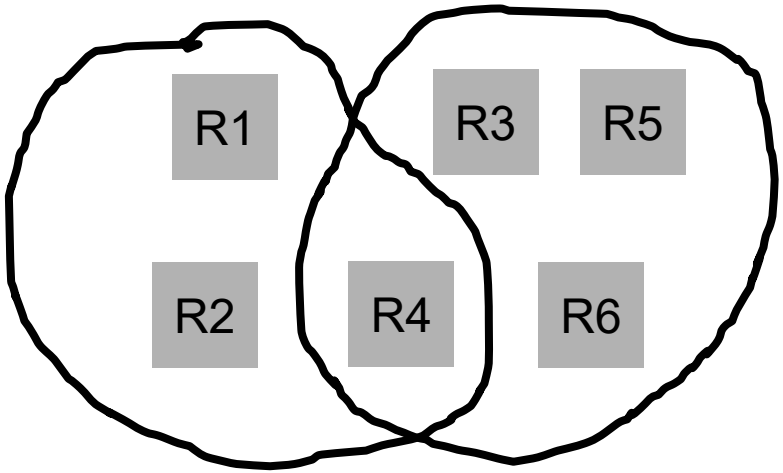
	Primary copy	Update everywhere
Synchronous (eager)	**Too expensive (usefulness?)**	**Too expensive (does not scale)**
Asynchronous (lazy)	**Feasible (limited scalability)**	**Feasible in many applications**

Alternative: Quorums

Quorum systems take a middle ground between the extremes of shipping all updates synchronously and asynchronously.

Updates are propagated asynchronously, but the original operation does not commit until a majority of replicas has acknowledged update success.

Reads can no longer contact a single replica to avoid stale reads – quorum sizes must be set in a way to preclude concurrent updates and to assert intersection of read and write quorums.



Quorums

For N replicas and read/write quorum sizes R/W respectively, the following conditions need to hold:

- No stale reads: R+W > N
- No concurrent updates: W> N/2

Quorum systems where one or both of these conditions are violated, have been referred to as “sloppy quorum“, e.g., Dynamo or Cassandra.

Quorum systems can be used to trade higher read latency for lower write latency (and vice versa).

Alternative: CRDTs

Idea: define distributed data types whose operations are commutative (e.g., add/subtract on a counter instead of set).

In the absence of updates, CRDTs are guaranteed to converge.

With sticky sessions, they often provide causal consistency.

Problems:

- Lack of implementations
- Long version histories
- Not generally applicable (e.g., no “database“ CRDT)

Summary so far

Replication is used for performance and fault-tolerance/availability purposes.

There are four possible strategies to implement replication solutions depending on whether it is synchronous or asynchronous, primary copy or update everywhere.

Each strategy has advantages and disadvantages (which are more or less obvious given the way they work).

There seems to be a trade-off between correctness (data consistency) and performance (throughput and response time).

REPLICA PLACEMENT

Introduction

Technische Universität Berlin

Replica placement is a major design issue. Permanent replicas can be

- within a LAN, or
- geo-distributed, that is, data is stored on servers in multiple geographic locations

In addition, server-initiated replicas (e.g., push cache) and client-initiated replicas (cache) may exist.

Due to higher latency in replica-to-replica communication, consistency-latency/availability tradeoffs become more pronounced in fog deployments.

State of the Art

Current systems do either not or only to a very limited degree consider geographic distribution and network topologies for replica placement and selection.

- No big deal in Cloud environments as nodes are close to each other and have high bandwidth connections
- Fog environments suffer from performance, QoS, consistency, and availability issues if distribution is unfavorable

State of the Art (cont.)

RDBMS replication and replication strategies do not work in the Fog, i.e.,

- Full replication in master-slave/quorum-based setups
- Sharding based on data item characteristics

Literature distinguishes four replica placement strategies:

- *Global mapping*
- *Hashing*
- *Chaining*
- *Scattering*

=> Are they a viable option for fog data management?

Global Mapping

In global mapping, storage systems control replica placement in a single centralized component.

PRO:

Supports arbitrary complex (and intelligent) replica placement decisions

CONTRA:

Comes with natural scalability and availability challenges

- Single point of failure
- All control flow needs to pass through a centralized component
- …

Example: Google File System (GFS)

In GFS, a single master server handles replica placement and selection across the entire (non-geo-distributed) GFS cluster; shadow master servers help to improve availability.

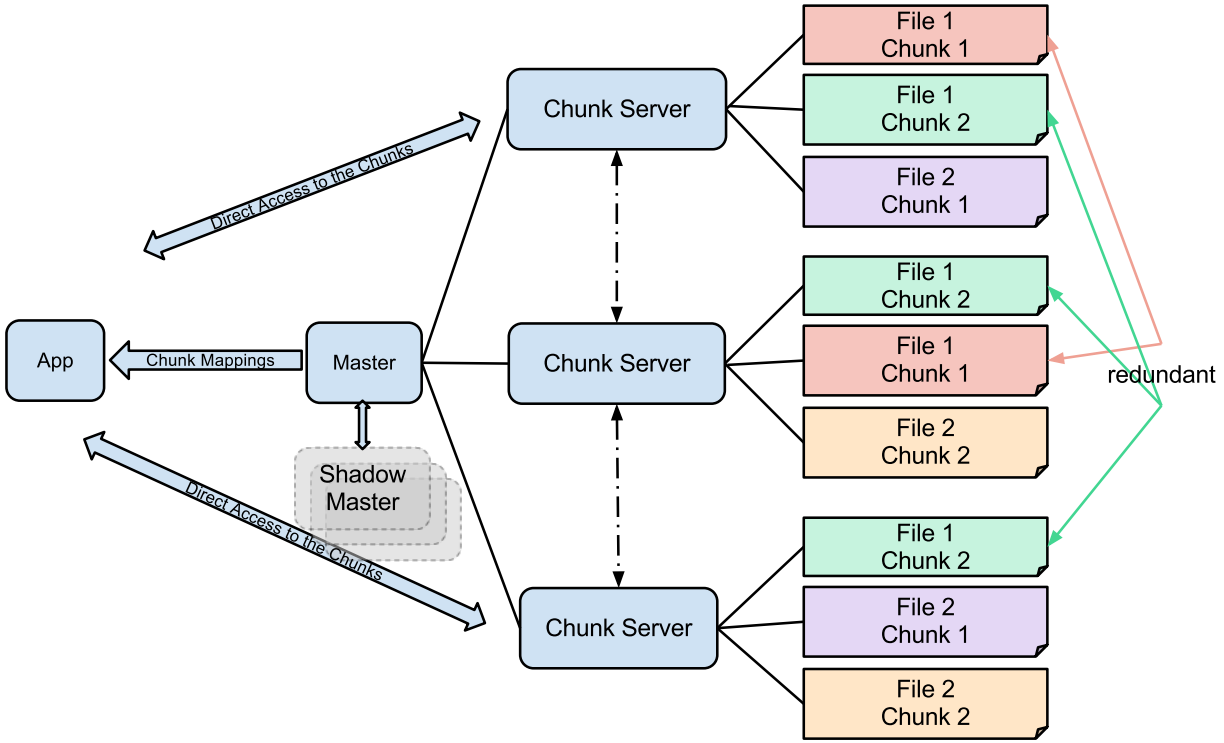


Image: https://upload.wikimedia.org/wikipedia/commons/c/c3/GoogleFileSystemGFS.svg

Example: Nebula

Nebula is a grid-inspired distributed edge store that centrally controls data placement in the DataStore Master. This master handles all aspects of data placement and replica selection.

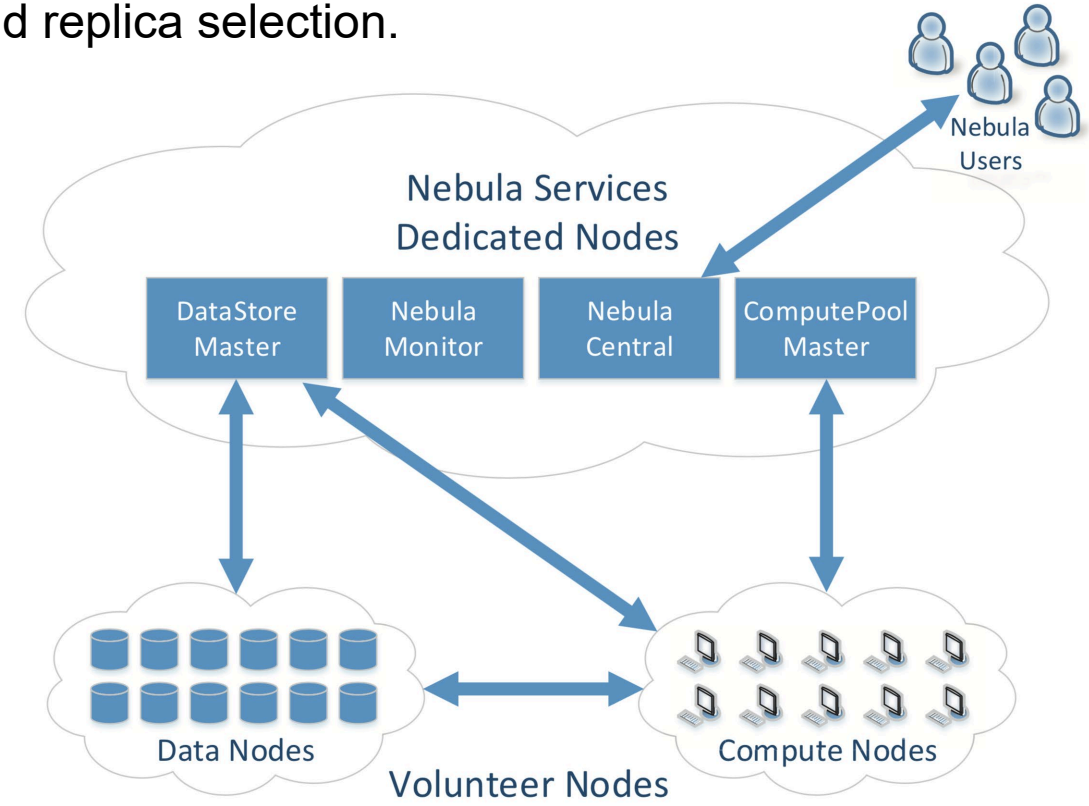


Image: Nebula: Distributed Edge Cloud for Data IntensiveComputing (Ryden et al.)

Hashing

In hashing-based replica placement, a hash value – usually of the data item's key – is used to deterministically identify a set of machines which will then store the corresponding data item.

PRO:

Scales very well as replica placement and selection are decentralized

CONTRA:

Does not cope well with high node churn rates

Not a good fit for Fog deployments as the full determinism of the static hash function …

- … makes it hard to consider underlying network topologies in replica placement
- … does not allow to place data close to actual access locations based on current demand

Example: Chord

Chord uses a consistent hash function to assign nodes and data items an m-bit ID. The IDs are arranged as a circle modulo 2^m, from 0 to $2^m - 1$. A data item is stored at the node whose ID is greater or equal to its own ID. Each nodes holds a pointer to is pre- and successor which are used to lookup data.

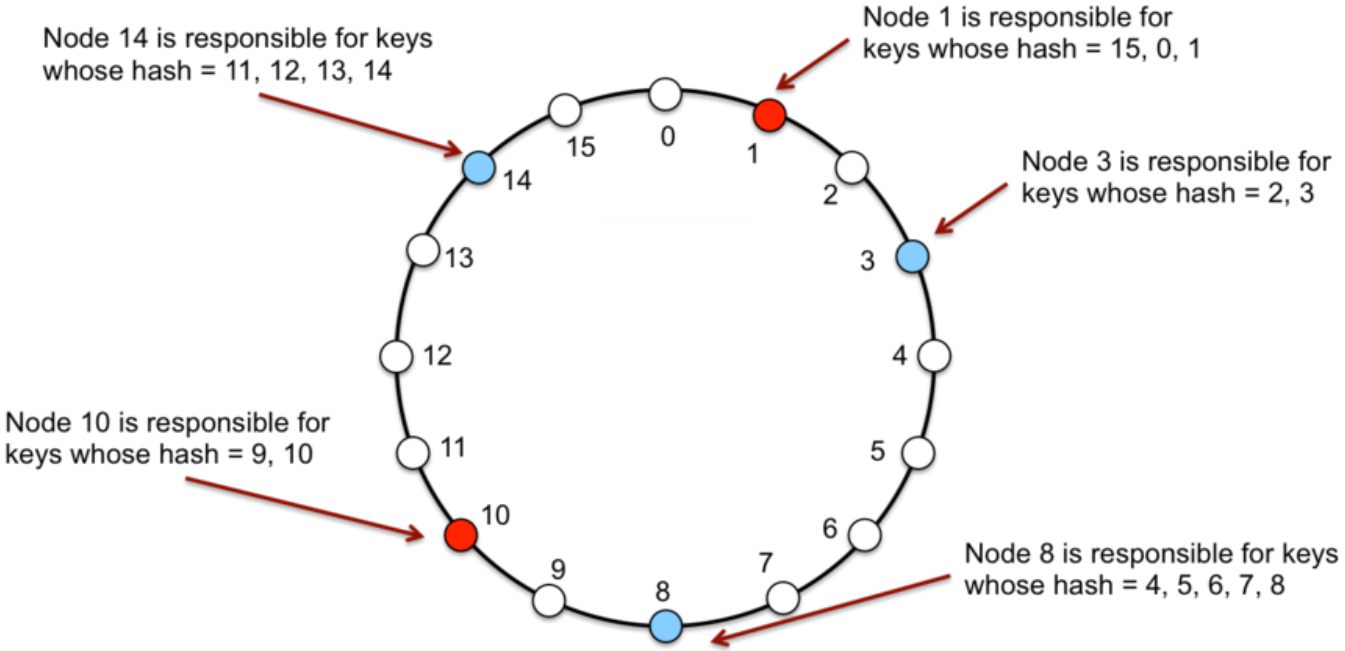


Image: https://www.cs.rutgers.edu/~pxk/417/notes/images/dht-dynamo-vnode.png

Other Examples

- PAST
- Pastry
- Kademlia
- Oceanstore
- Dynamo
- Cassandra
- Voldemort
- Riak
- …

Chaining

In chaining-based replica placement, additional replicas are created (deterministically) on adjacent machines of a primary replica selected through some other replica placement strategy.

PRO:

Makes it possible to control where chaining replicas should reside

CONTRA:

- Tends to clutter replicas in close physical proximity
- Is relatively static, so not equipped well for dynamic replica movement

Example: Dynamo

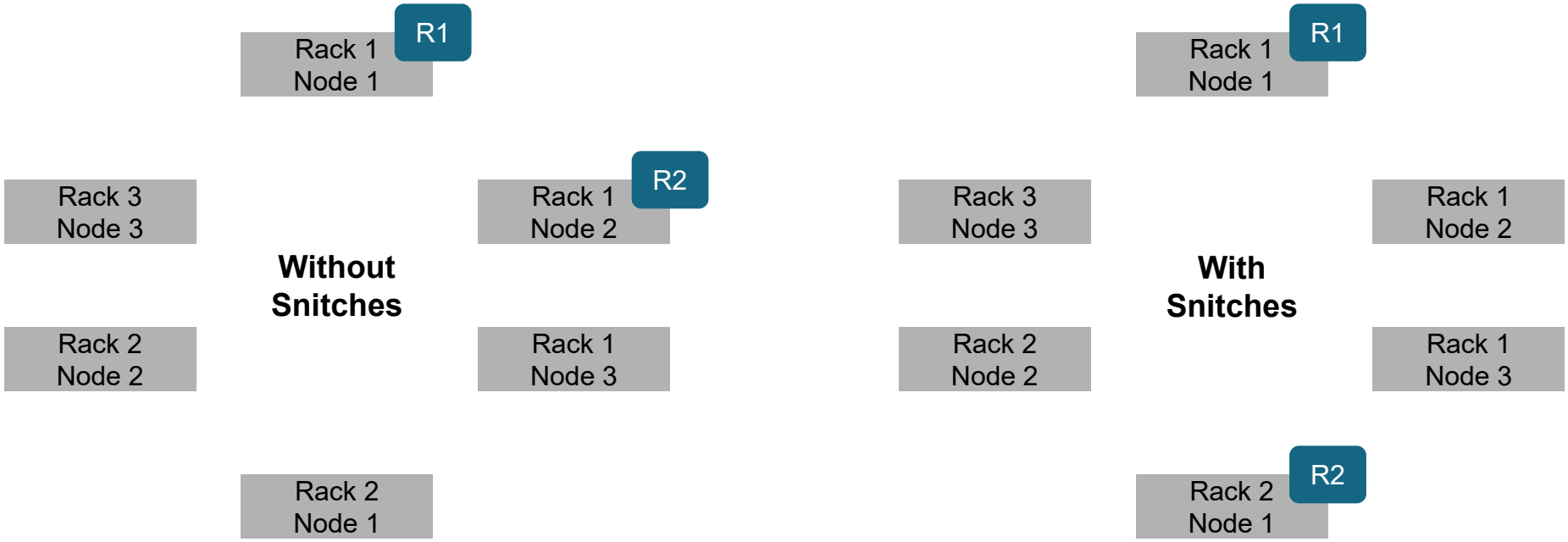
Dynamo selects a primary replica through consistent hashing, and then places an additional replica on the next N-1 nodes on the ring as defined by the replication factor.

In the presence of temporary node failures, Dynamo uses a slightly relaxed version of consistent hashing with chaining.

In that case, the first N healthy nodes become replicas starting from the keyrange identified for the primary replica until the first N nodes are available again. => hinted handoffs

Example: Cassandra

Cassandra has a feature called “snitches” to prevent replicas being stored on machines in the same rack, or in the same data center.



Scattering

Scattering creates a pseudorandomized but deterministic distribution of replicas across machines.

Example: CRUSH algorithm which computes a pseudorandom data placement distribution based on a hierarchical description of the target cluster.

PRO:

Can be used for good placement in geo-distributed deployments.

CONTRA:

Poorly equipped to deal with end user mobility and resulting access pattern variance across system nodes

Hybrid Approaches might be the way to go

All four replica placement strategies are not a natural fit for the Fog.

Hybrid approaches, that use a combination of different strategies, might be the way to go.

Examples include:

- PNUTS: uses global mapping for replica placement within a region and full replication across regions
- DynamoDB: offers global mapping for cross-region replication
- FARSITE: combines global mapping with scattering
- IPFS: uses a BitTorrent-inspired protocol for data exchange and a hashing algorithm to determine storage locations
- …

Example: PNUTS

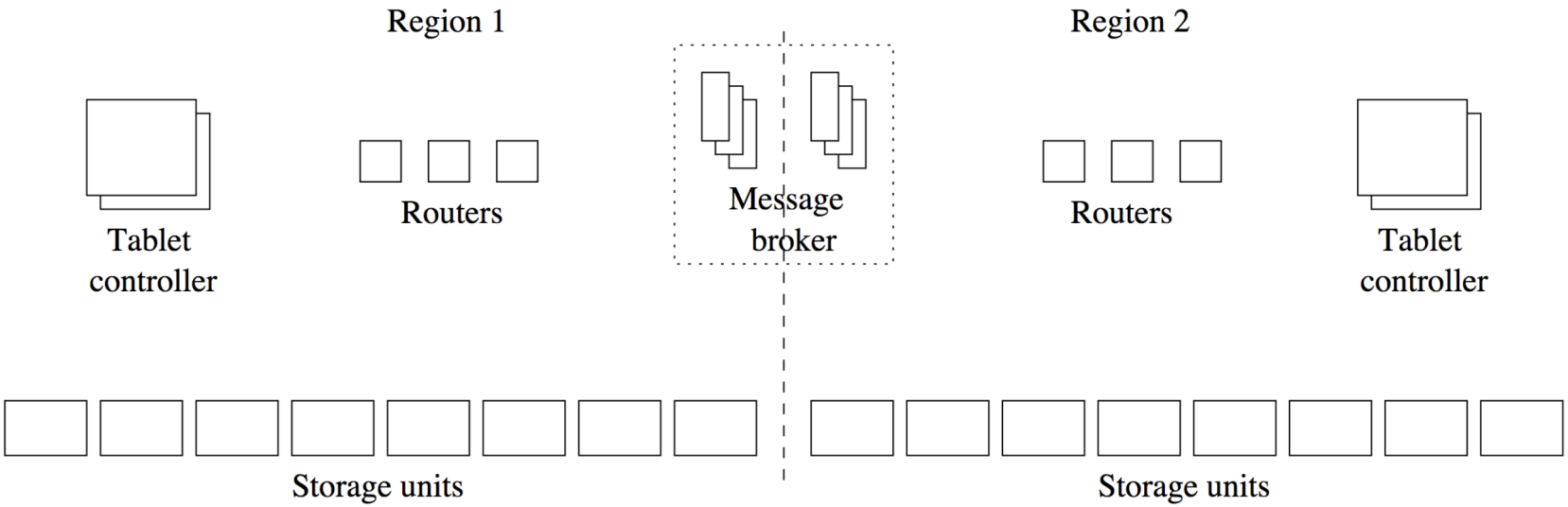


Image: Cooper et al. (2015) PNUTS: Yahoo!'s Hosted Data Serving Platform

Example: DynamoDB

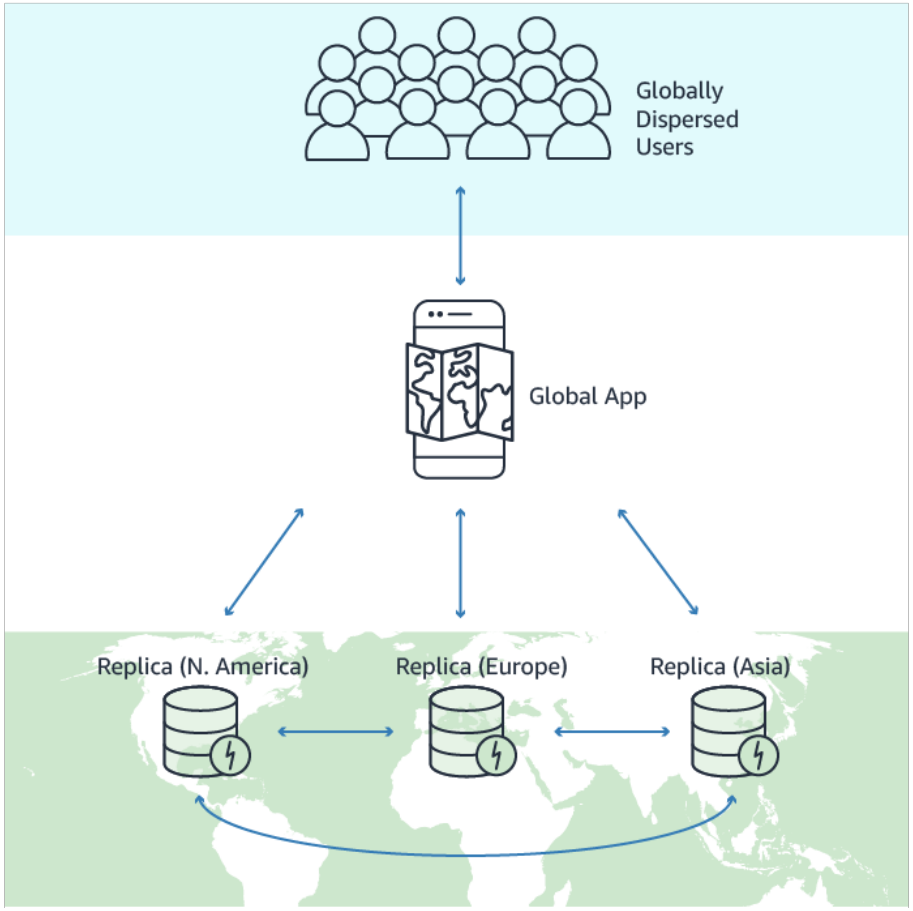


Image: https://d1.awsstatic.com/product-marketing/DynamoDB/DynamoDB_Global-Tables-02.6f8a0376e5e828b2cbc3247c1b93350e8f31141e.png

Fog data management

EXAMPLE: IPFS + ROZOFS

Confais et al. An Object Store Service for a Fog/Edge Computing Infrastructure Based on IPFS and a Scale-Out NAS. IEEE ICFEC 2017.

IPFS is a Peer-to-peer Distributed File System

- As IPFS is peer-to-peer, no nodes are privileged and all IPFS nodes stored IPFS objects in local storage
- Nodes connect to each other to transfer objects. Objects can be any data structure such as files or directories
- Objects are not identified by their location (such as an URL/IP address), **but by their content** (in form of a hash)

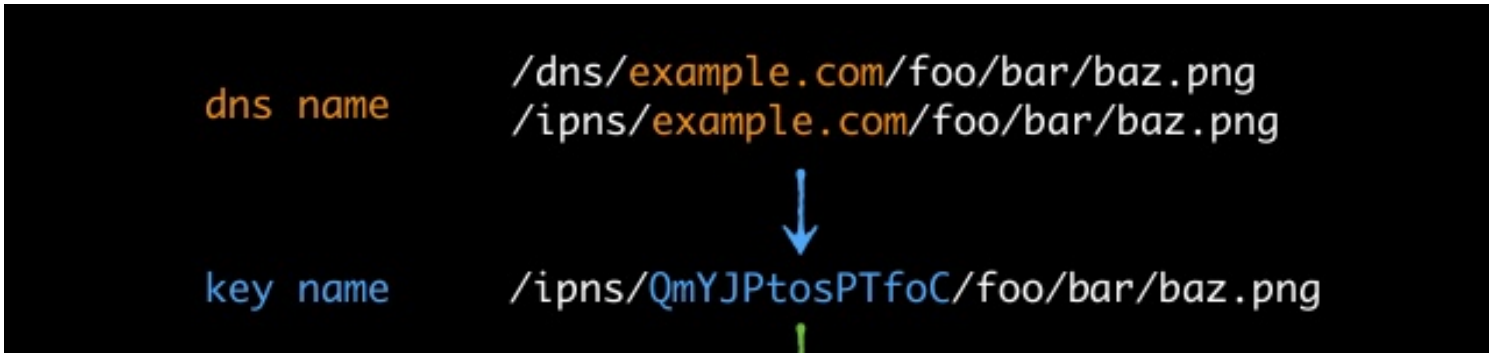


Image: https://hackernoon.com/ipfs-and-merkle-forest-a6b7f15f3537

➢ Main Idea: Modeling all Data as part of the same Merkle DAG

Benefits of Using Merkle Trees

Links between two nodes are in the form of cryptographic hashes. This is made possible by the Merkle DAG data structure. Advantages of this approach include:

Content-based Addressing

- All content is uniquely identified by its cryptographic hash (including links)
- Sharing the address (hash) of a root directory is enough to allow another person to find all related data

Tamper-proof

- All content is verified with its checksum.
- Tampered or corrupted data is detected by IPFS as the hash changes

No duplication

- Objects that hold the exact same content are equal
- IPFS stores these only once

IPFS – Enabling Technologies & Core Concepts

- Distributed Hash Tables
- Block Exchange - Bit Torrent
- Version Control Systems – Git
- Self-Certifying Filesystems

=> IPFS is not quite fog-ready. Can we adapt it?

Why is IPFS not Fog Ready?

Using a DHT for metadata management has the downside that each object access involves the DHT to locate the object if it is not available on the requested node.

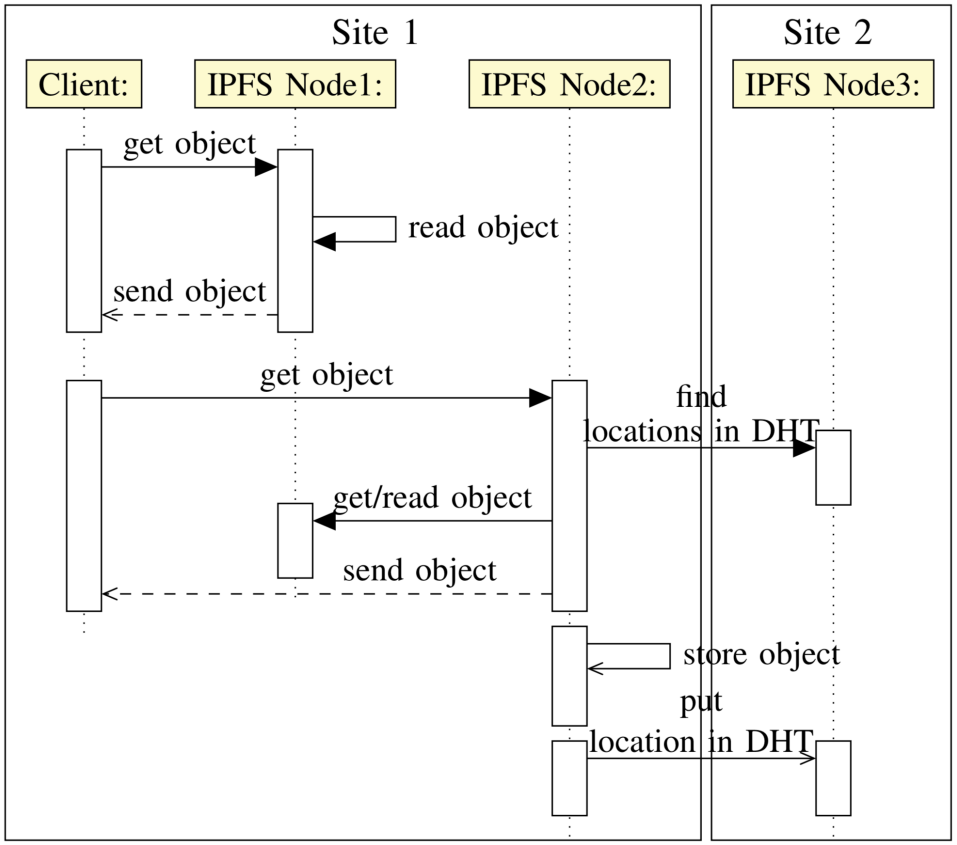


Image: Confais et al. (2017) An Object Store Service for a Fog/Edge Computing Infrastructure Based on IPFS and a Scale-Out NAS

Taking Advantage of the Physical Topology

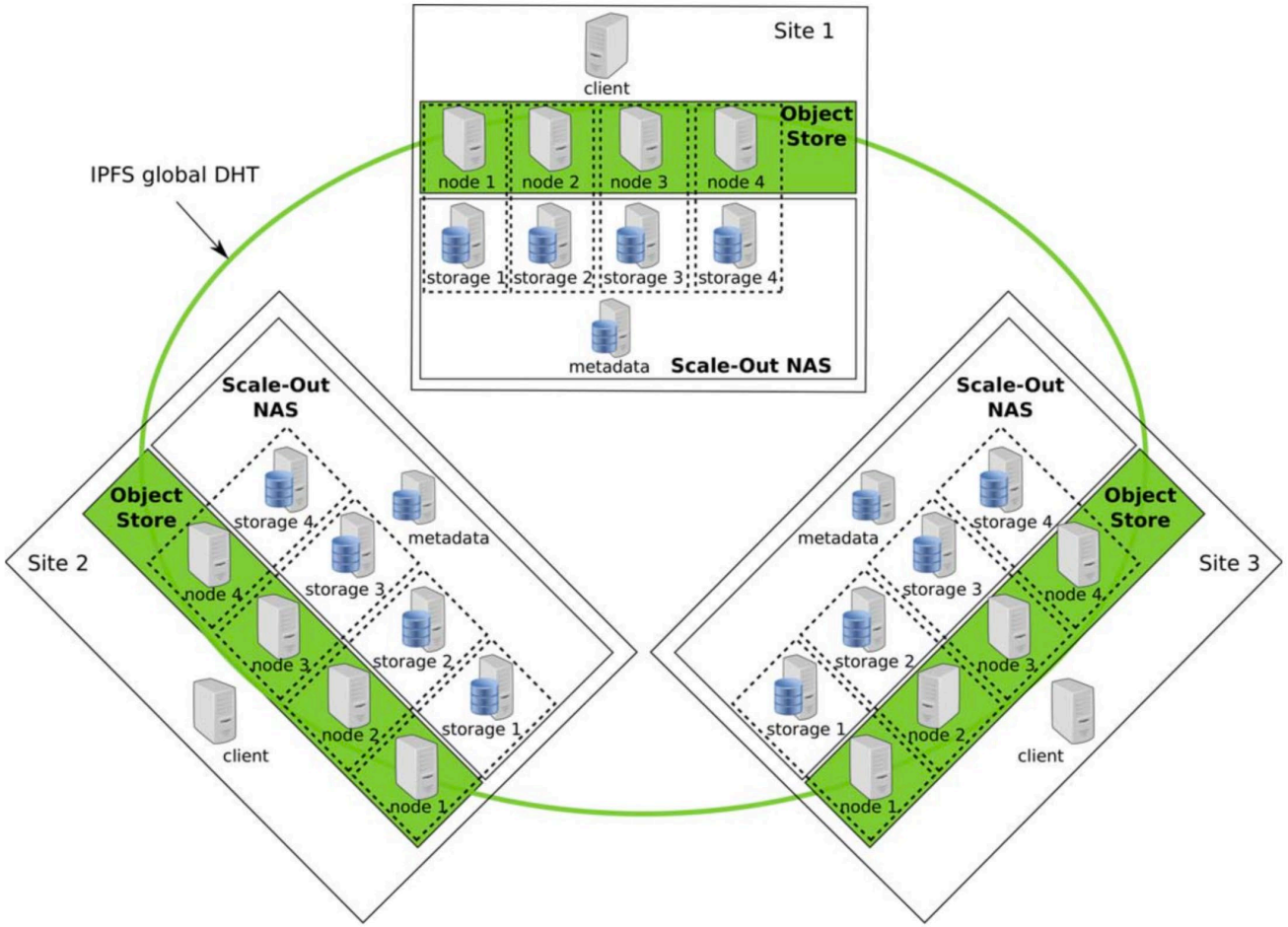


Image: Confais et al. (2017) An Object Store Service for a Fog/Edge Computing Infrastructure Based on IPFS and a Scale-Out NAS

Adding a Scale-Out NAS Enables on Site Reads without Using the DHT

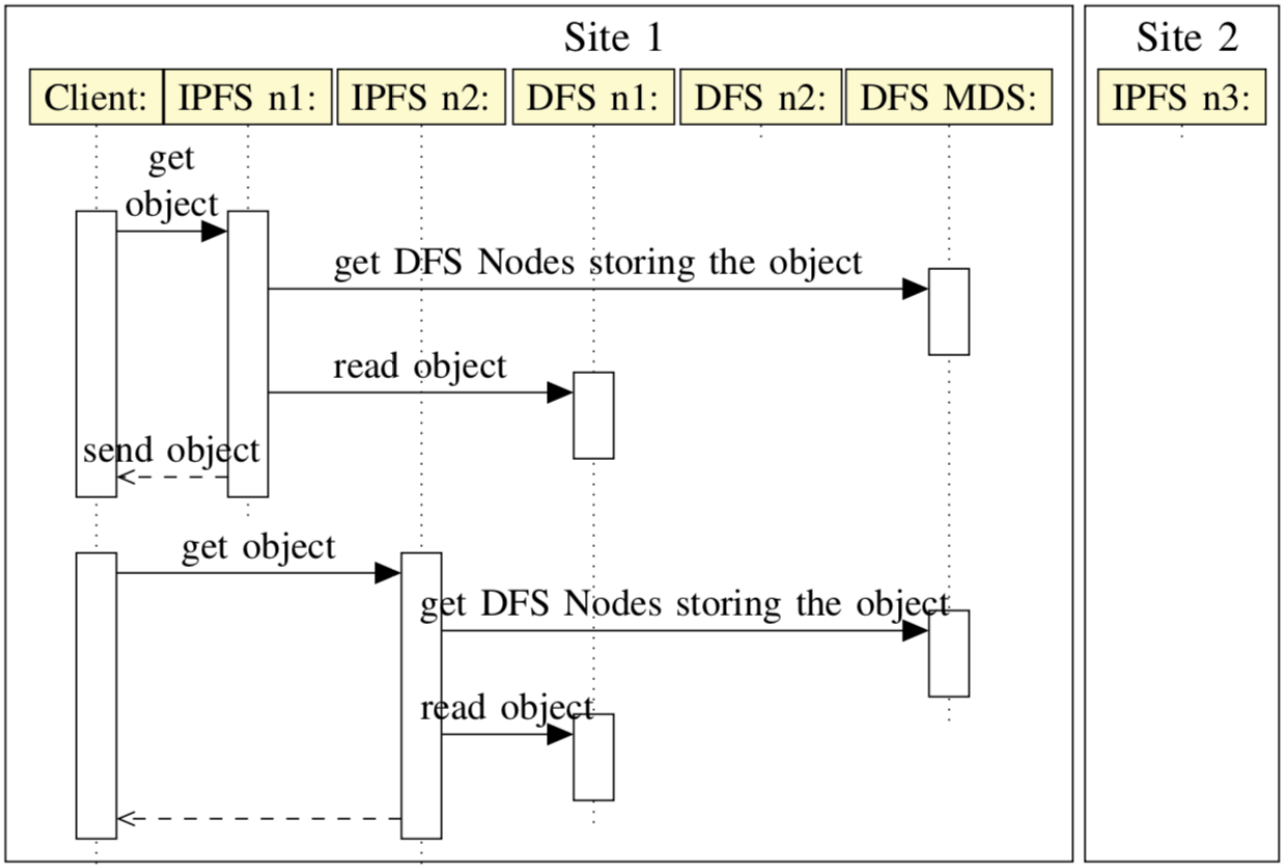


Image: Confais et al. (2017) An Object Store Service for a Fog/Edge Computing Infrastructure Based on IPFS and a Scale-Out NAS

Technische Universität Berlin

Fog Data Management

GLOBAL DATA PLANE

Zhang et al. The cloud is not enough: Saving iot from the cloud. USENIX HotCloud 2015.

Mobile Cloud Computing

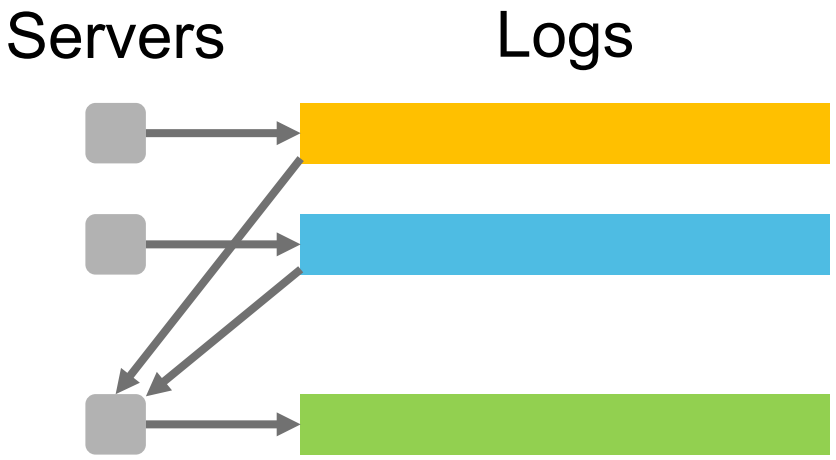
EINSTEIN CENTER Digital Future

The Global Data Plane (GDP)

The GDP is a data-centric abstraction focused around the distribution, preservation, and protection of information.

GDP builds upon append-only, single-writer logs:

- lightweight and durable
- multiple simultaneous readers
- no fixed location, migrated as necessary
- compositions are achieved by subscriptions



The GDP Operates Above the Network Level

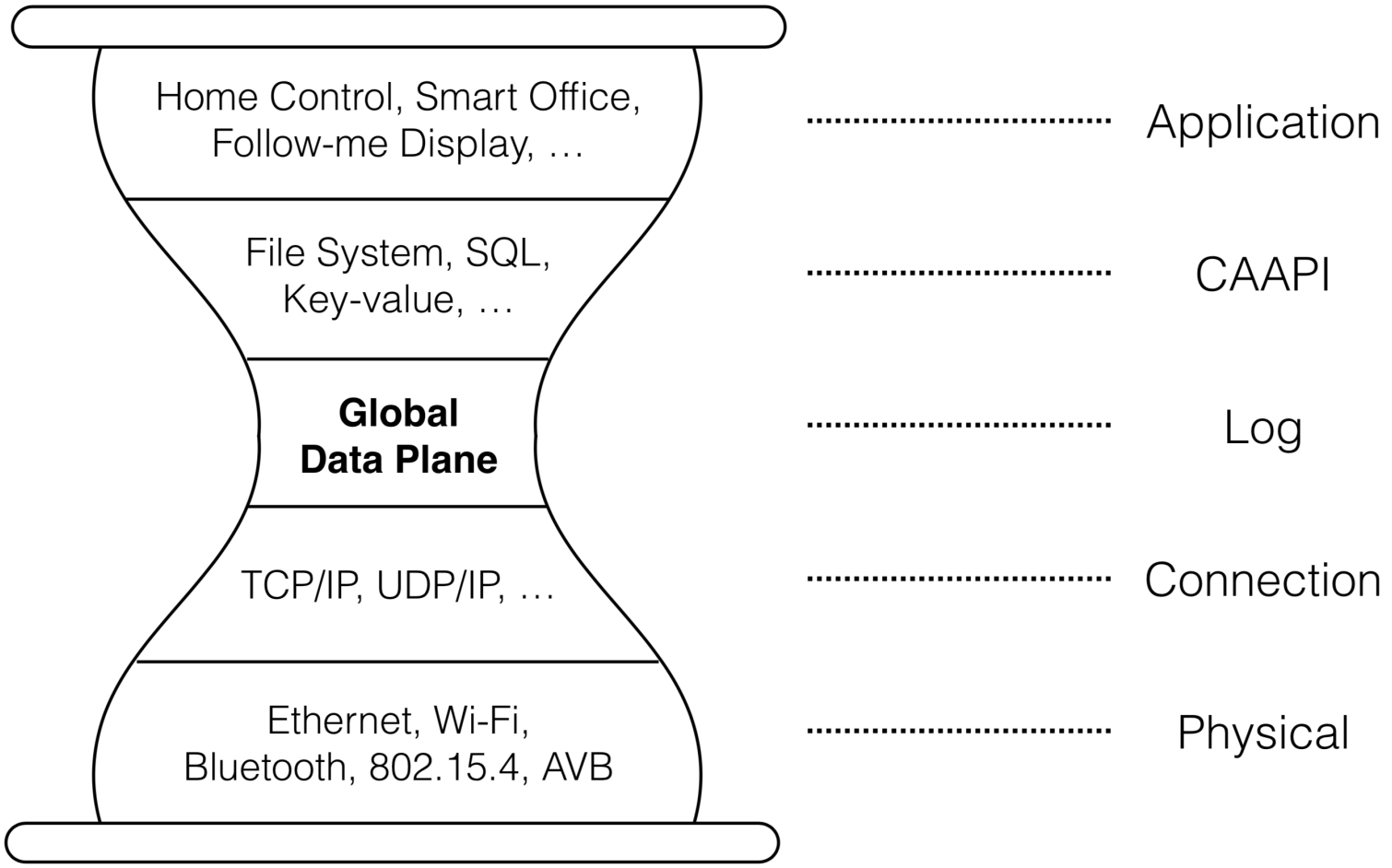


Image: Zhang et al. (2015) The Cloud is Not Enough

Location-Independent Routing

- GDP employs a location-independent routing in a large 256-bit address space
- Packages are routed through an overlay network that uses Distributed Hash Tables which enables flexible placement, controllable replication and simple migration of logs
- GDP places logs within the infrastructure and advertises the location to the underlying routing layer
- Placement (and replication) of logs can be optimized for latency, QoS, privacy, durability, …
- Logs themselves are split into chunks whose placement can be optimized for durability and performance

Technische Universität Berlin

Fog Data Management

FBASE / FRED

Hasenburg et al. Towards a Replication Service for Data-Intensive Fog Applications. ACM SAC 2020.

Mobile Cloud Computing

EINSTEIN CENTER Digital Future

Application-controlled replica placement

Let applications (or higher-level layers) decide upon replica placement and expose interface. FBase/FReD* handles everything else.

Key abstractions:
- Nodes
- Keygroups
- Keygroup members

* FBase = Java/Kotlin prototype, FReD = Go prototype

Nodes

Nodes are a group of one or more machines within one geographical site including a hosted or embedded storage system.

Examples:
- “Small“ node: Raspberry Pi with embedded BerkeleyDB
- “Large“ node: 10 EC2 instances with DynamoDB

Nodes only interact with other nodes as a whole but not with their individual machines (even though work is distributed within nodes).

Coordination within nodes is done through the storage system.

Address infrastructure heterogeneity (to some degree)

Keygroups

A keygroup is a group of data items that are replicated together.

Each keygroup has its own ACL.

Applications declaratively specify the set of keygroup members which controls data distribution. Updates can be made at runtime.

Communication of keygroup members is based on pub/sub and handles data distribution automatically.

➡ Let applications declaratively define data flows

Keygroup members

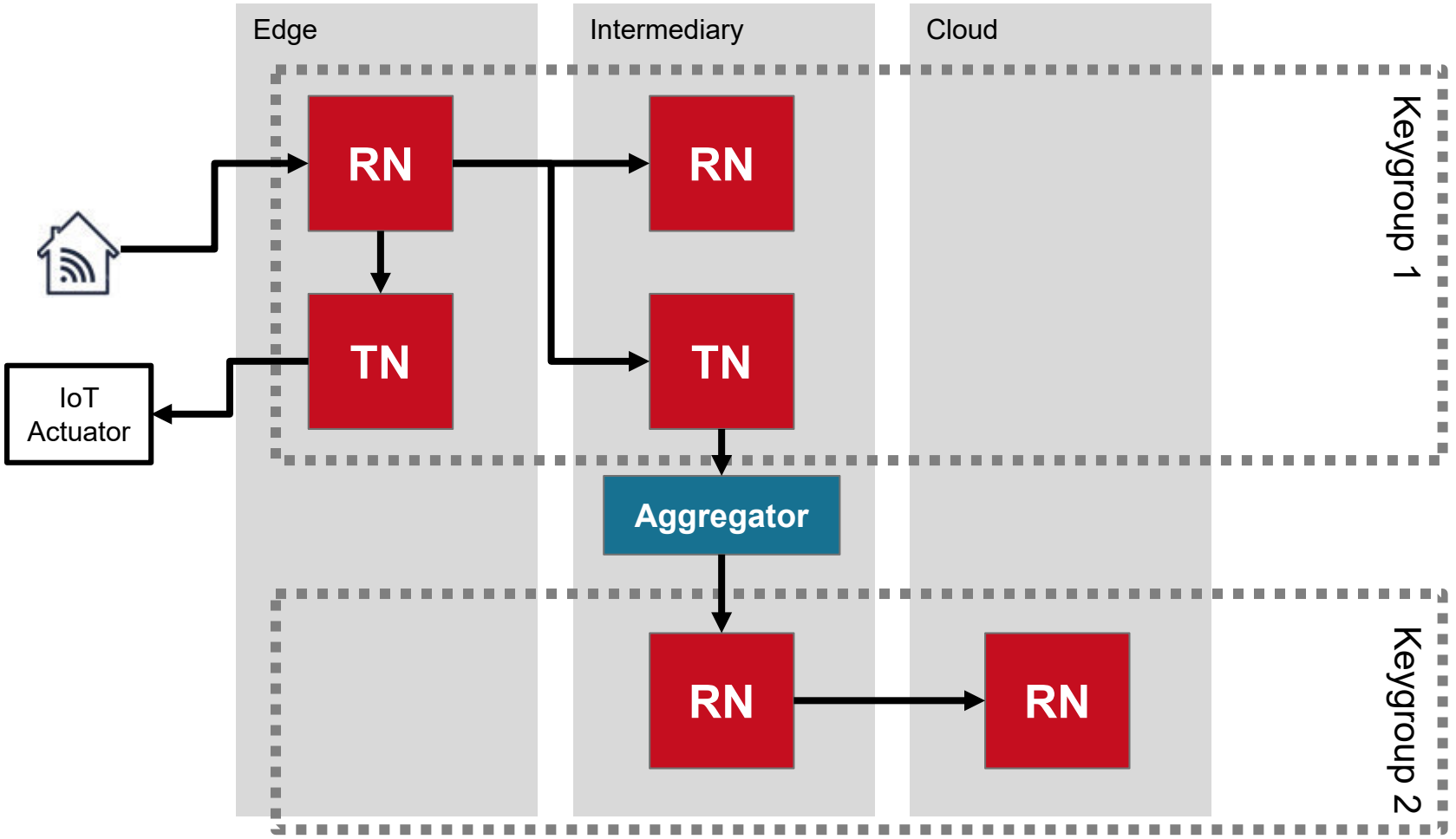
Nodes can be a keygroup member in one or both roles:

- Replica nodes store a data replica, serve client requests, and manage keygroup configuration
- Trigger nodes receive all updates as a stream of events and may trigger external systems via an event-based interface

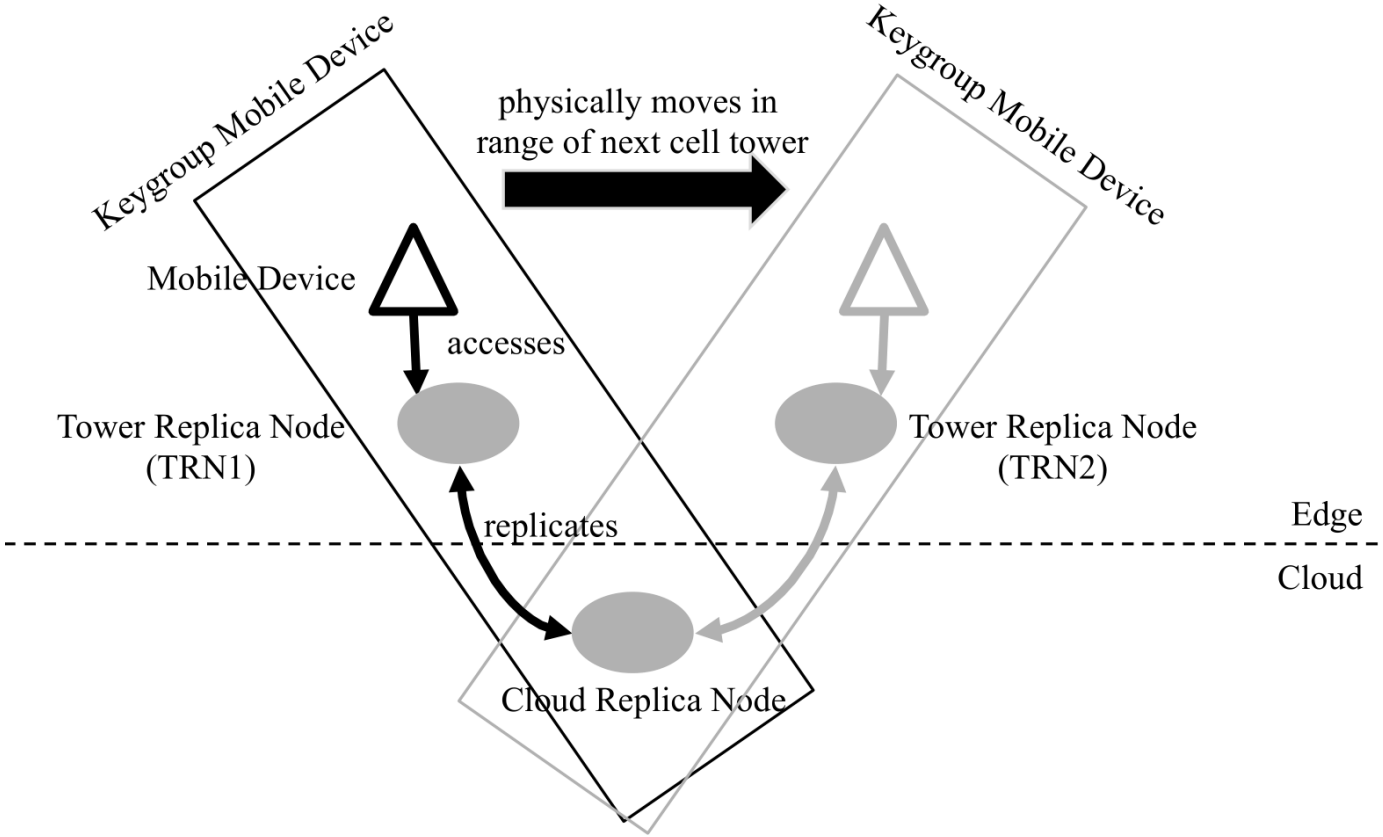
Applications can specify a TTL for data retention on replica nodes.

Support data transformation and integration of external systems

Example


Edge
Intermediary
Cloud
RN
RN
TN
TN
IoT
Actuator
Aggregator
RN
RN
Keygroup 1
Keygroup 2

Example: mobile app



Example: mobile app (cont.)

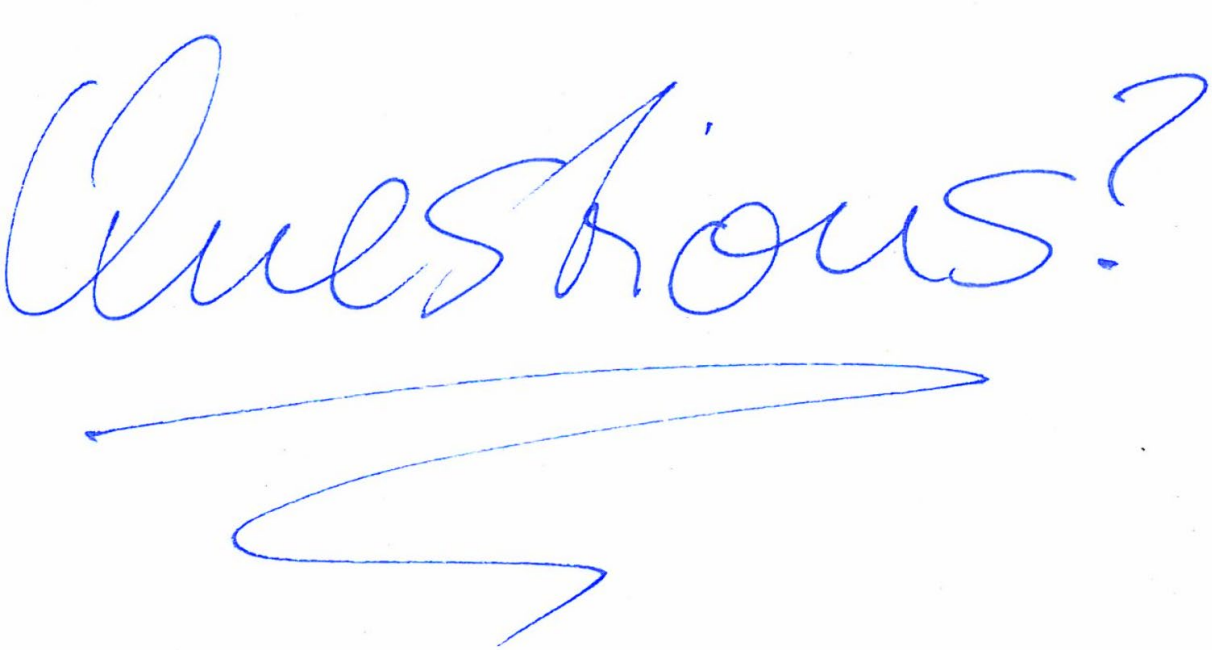
```
fun setupKeygroup() {
    addToKeygroup(keygroupId = "mobile-app/mobile-device/data",
            clientIds = listOf("Mobile Device"),
            replicaNodeIds = listOf("TRN1", "Cloud Replica Node"))
}

fun onMovement() {
    removeFromKeygroup(keygroupId = "mobile-app/mobile-device/data",
            replicaNodeIds = listOf("TRN1"))

    addToKeygroup(keygroupId = "mobile-app/mobile-device/data",
            replicaNodeIds = listOf("TRN2"))
}
```

Summary

Replication is a key challenge in fog computing

Different replication and replica placement strategies exist

First steps towards integrated fog data management (but we‘re not there yet)


Questions?