

263-3855-00: Cloud Computing Architecture

Section 07: Storage Systems

Swiss Federal Institute of Technology Zurich (ETH Zürich)

Last Edit Date: 05/31/2025

Disclaimer and Term of Use:

1. We do not guarantee the accuracy and completeness of the summary content. Some of the course material may not be included, and some of the content in the summary may not be correct. You should use this file properly and legally. We are not responsible for any results from using this file.
2. This personal note is adapted (derived) from the slides and codes from *Professor Ana Klimovic and people from ETH Zurich Systems Group*. Please [contact us](#) to delete this file if you think your rights have been violated.
3. This work is licensed under a [Creative Commons Attribution 4.0 International License](#).

Storage data in the cloud poses unique challenges:

- Cannot be based on local disks of the computing machines
- Must scale to very large sizes
- Must provide reliability and availability at a reasonable cost
- Must work with existing applications
- Should take advantages of cloud services and cloud options

The most distinct aspect of data storage in the cloud is the clear separation between the storage and the compute layers:

- Allows to scale out both independently
- Does not tie applications to a particular location
- It adds latency as it involves network transfers

Given its scale, a cloud cluster see many problems and failures over time as well as transient failures:

- Hardware faults (machines, disks, RAM, components)
- Infrastructure failures (DBS, network, misconfigurations, power supply)
- Environment (earthquakes, fires, power supply failures)
- Planned events (upgrades, updates, patches)

Data must still be available. Having the data in a single device would just not work as it would make the cloud very unreliable

Heavy use of data replication at all levels, performance and cost

I. Performance vs. consistency

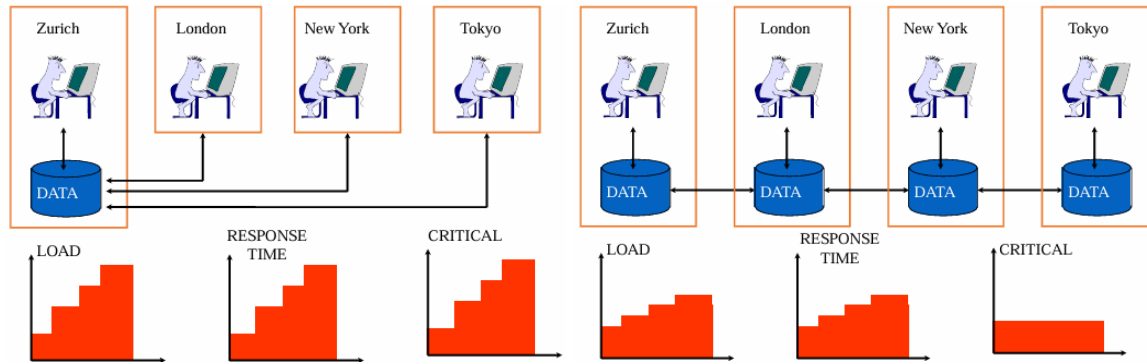
1. Data replication

Replication is a common strategy in data management: RAID technology (Redundant Array of Independent Disks), Mirror sites for web pages, backup mechanisms (1-safe, 2-safe, hot/cold standby)

Why replication?

- **Performance:** Location transparency is difficult to achieve in a distributed environment. Local accesses are fast, remote accesses are slow. If everything is local, then all accesses should be fast.
- **Fault tolerance:** Failure resilience is also difficult to achieve. If a site fails, the data it contains becomes unavailable. By keeping several copies of the data at different sites, single site failures should not affect the overall availability.
- **Application type:** Databases have always tried to separate queries from updates to avoid interference. This leads to two different application types - OLTP and OLAP - depending on whether they are update or read intensive.

Remote access to data vs. replication



2. Replication Strategies

When the updates are propagated?

	Synchronous (eager)	Asynchronous (lazy)
Description	Synchronous replication propagates any changes to the data <i>immediately</i> to all existing copies. Done in the context of transactions. ACID is applied to all copies of updates.	Asynchronous replication first executes the updating transaction on the <i>local copy</i> . Then the changes are propagated to all other copies, which are inconsistent (having different values) during propagation.
Advantages	- No inconsistencies (identical copies). - Reading the local copy yields the most up to date value. - Changes are atomic.	A transaction is always local (good response time).
Disadvantages	A transaction has to update all sites (longer execution time, worse response time).	- 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.

- Most database systems are using synchronous replication, although it is expensive in terms of latency, still over years shows that it is better than the alternatives.
- Asynchronous replication is used in systems that tolerate the lack of consistency: replication not necessarily used for increasing performance, applications where data can be easily partitioned or sharded, to implement backup style solutions. Common in many no-SQL systems.

Where the updates can take place?

	Update Everywhere (group)	Primary Copy (master)
Description	Changes can be initiated at <i>any</i> of the copies. Any of the sites which owns a copy can update the value of the data item.	Only <i>one copy</i> can be updated (the <i>master</i>), all others (secondary copies) are updated reflecting the changes to the master.
Advantages	- Any site can run a transaction. - Load is evenly distributed.	- No inter-site synchronization is necessary (it takes place at the primary copy). - There is always one site which has all the updates.
Disadvantages	Copies need to be synchronized.	- The load at the primary copy can be quite large. - Reading the local copy may not yield the most up to date value.

- Update everywhere is used in databases, file systems, and object stores. Quorum bases systems, meaning that $N < R + W$ and $N < W + W$, where R is read quorum, W is write quorum, and N is number of copies, which guarantees to always read the latest copy.
- Primary copy is used in databases such as Amazon Aurora and Microsoft Azure SQL, basically solutions where the copies are there to provide backups or higher availability.

Where vs. when

	Update Everywhere	Primary Copy
Synchronous	Advantages: - No inconsistencies. - Elegant (symmetrical solution). Disadvantages: - Long response times. - Updates need to be coordinated.	Advantages: - Update not coordinated. - No inconsistencies. Disadvantages: - Longest response time. - Only useful with few updates. - Local copies can only be read.

Asynchronous	Advantages: - No centralized coordination. - Shortest response time. Disadvantages: - Inconsistencies. - Update can be lost (reconciliation).	Advantages: - No coordination necessary. - Short response times. Disadvantages: - Local copies are not up to date. - Inconsistencies.
---------------------	---	---

3. CAP theorem

The CAP theorem is basically an impossibility triangle: a system can only have 2 of these three properties.

- **Atomic Consistency:** At point in time, the same request to any server returns the same result, in other words, all nodes see the same data.
- **Availability:** The system is available for request at all times (SLA with very high availability).
- **Partition tolerance:** The system continues to function even if the network linking its machines is occasionally partitioned.

It is a complex trade-off involving performance and cost. In practice, availability has many aspects. For example, a slow system might be available but it could be useless.

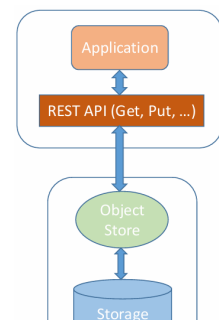
II. Storage systems & cloud native databases

1. Storage Systems

- **Block storage:** Mirrors the interface of a local disk, access by blocks / pages, application manages the data, low level interface.
- **File storage:** On top of block based devices but interface to the user is a collection of blocks called a file, which maintains some metadata about what is stored (file name, owner, size, creation time, etc.), file system interface.
- **Object storage:** On top of file systems, objects consist of data and metadata plus system identifier, REST API (GET, PUT, DELETE).
- **Key-value store:** Low latency access to small data items, identified through a key, retrieves the entire value.
- **Databases:** Structured and declarative access to data organized with a schema, interface is SQL.

2. Object storage

- A very famous example is Amazon S3, which has an interface as a REST API
 - GET, PUT, DELETE, etc.



- Objects are immutable (cannot be modified) and have a lot of metadata attached to them (customizable)
- Very high availability through automatic replication across many systems
- Object store supports many features, including partial processing of the data for specific formats (S3 select) and even adding using defined functions (S3 object lambda)
- Not intended to store files as objects (can be done, though) as the interface provided is different and requires changing applications.
- Object storage is the cloud native storage solution (blocks and files are “low level” and used for compatibility reasons as most systems are designed in those terms)
- Naming is based on URLs as interface is HTTP with three levels - account, containers, and blobs
- In Azure storage
 - Block blobs: for text and binary data, divided into blocks that can be accessed individually
 - Append blobs: like block blobs but optimized for append operations (e.g., implementing logs)
 - Page blobs: 512-bytes pages that can be indexed (used for databases)

3. Key value store

The cloud provider and the user have an interest in cheaper storage (actual storage and access cost). But cheap storage is slow, so system designers prefer to have faster storage, which is more expensive. This becomes a major problem for systems processing large amounts of data. A way to address the situation is to use intermediate caches - cloud provided caches and key value stores.

- Blocks, files, and objects are pure storage based on a somewhat similar notion, but each one of them adds more information and more functionality at the price of latency.
- Internet scale created a demand for something slightly different:
 - Access a chunk of data that does not need to be parsed (unlike objects) but can be of various sizes (unlike blocks) and needs to be system supported names that simplifies finding the data (unlike file systems)
 - Data can be dynamic with many entries being created and deleted (not good for blocks, files, or objects)
- The solution is key value stores (KVS)

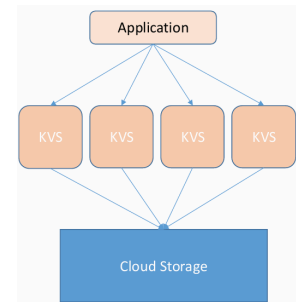
KVS architecture

- Data stored as a key-value pair
 - Key: identifier used to locate and retrieve the data
 - Value: a binary object of arbitrary size

- Data organized as a distributed hash table
 - Hash the key, then get the location for the value
- KV pairs stored with several copies in several places for availability
- Access if quorum based for consistency
- Typically, eventual consistency
- Actual storage uses already reserved buckets of different sizes and objects are located in the corresponding baskets
- Can be run in main memory for very fast access or with values in storage for large data items

KVS as caches

- KVS are often used as main memory caches to avoid having to access cloud storage:
 - Higher throughput
 - Lower latency
 - Databases: store results of a query in a value, key is generated by hashing the query. When running queries, check first the KVS and answer from there if the query is stored
 - Web pages, pictures, ads, etc.
- A very common use case for memcached



4. Databases

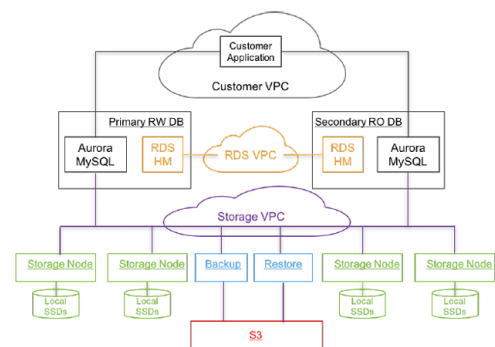
- Databases provide data storage but also data management and the ability to process the data
 - Traditional databases: run on a VM using block storage, “own” the data
 - Cloud native databases: run on VM using different forms of storage and cloud services, “own” the data
 - Databases as a service: run on top of object storage but do not store data themselves, they use the object storage as input
- The cloud has created the opportunity to specialize databases to concrete applications, leading to an explosion in the available systems and their features

The problem with traditional database engines in the cloud

- In the cloud, the separation of storage and compute causes overhead if one is not careful
- A traditional database generates too much traffic in terms of I/O, especially if it is mirrored for availability

Improvement: Amazon Aurora

- A modification of MySQL to make it work in the cloud



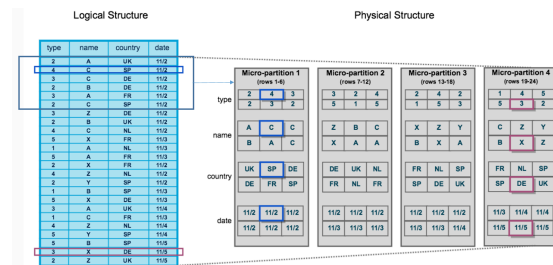
- Separate storage and cache management from transaction processing
- Primary copy, asynchronous replication
- 6 copies of the data in 3 different availability zones, continuous backup on S3
- Updates with quorum (4 out of 6)
- Primary copy and up to 15 read-only replicas (20 ms lag)

Innovation: Snowflake

- A data warehouse specialized for analytical queries developed entirely on the cloud (cloud native)
- Separates compute (nodes running VMs with a local disk) from storage (Amazon S3)
- A virtual warehouse is a collection of worker nodes (EC2 instances in Amazon)
- Each worker node has a cache in its local disk where it stores the objects (table files or parts thereof) accessed before
- The cache uses a simple LRU replacement policy

Micro-partitions

- Micro-partitions are Snowflake's name for extents. What is interesting is how they are organized to facilitate query processing:
- Size ranges between 50 and 500 MB (before compression, the data is always compressed when in S3)
- Each micro partition has metadata describing what is inside
- The metadata can be read without reading the whole micro-partition
- The metadata is used to read just the part of the micro-partition that is relevant
- Data in the micro-partition that is relevant
- Data in the micro-partition is stored in columnar form (by columns not by rows)



III. Caching & CDNs - accelerating cloud storage

1. What are CDNs / edge computing?

Simply speaking, having more, smaller data centers.

Advantages:

- Lower latency to client
- Cheaper bandwidth to client, since it costs less to transmit data over shorter distances

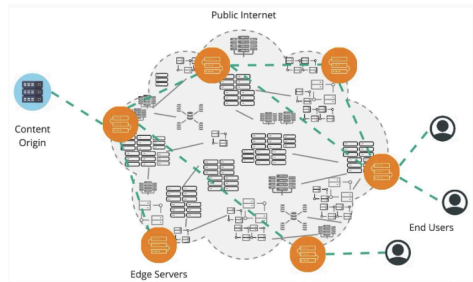
Disadvantages:

- Less data storage capacity per location. Desired data might not be available

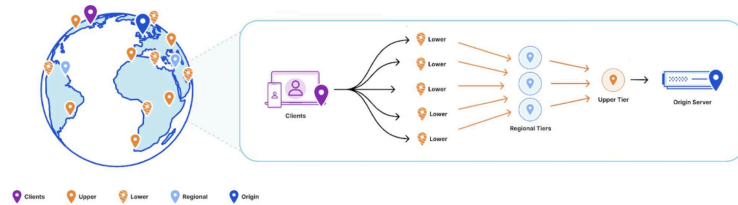
- Less compute and memory capacity per location. Might have to redirect to another nearby location
- Harder to manage.

Varied CDN designs:

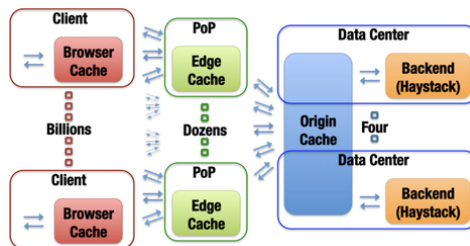
- **Highly distributed CDNs**, for example, Akamai with 1000+ locations



- **Datacenter-based CDNs**, for example, Cloudflare and Fastly



- **Specialized CDNs**, for example, Meta photo CDN and Netflix CDN

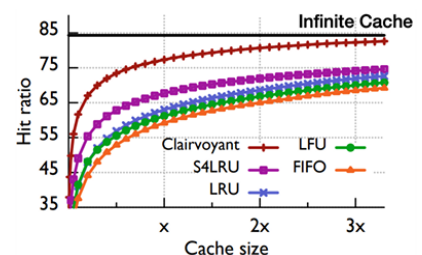


2. Other uses for the edge:

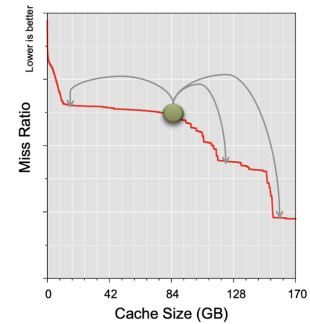
- Edge storage: In addition to a common “static file” served from CDNs is stored video (e.g., YouTube, Netflix). They are large and frequently accessed, thus, easy to cache.
- Edge computing is an emerging trend: Cloud gaming puts a lot of compute on a remote server. Latency requirements force the compute to be close to the user. RANs in 5G networks.

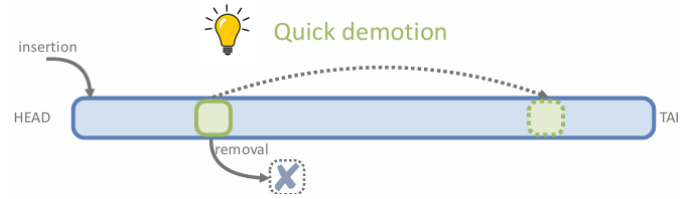
3. Hit ratio algorithms

- Increase cache size: a useful tool - Miss Ratio Curve (MRC)
- Advanced cache algorithms
 - Different caching algorithms
 - Least Recently Used (LRU)



-





LRU vs. FIFO vs. SIEVE

		Efficiency	Scalability
	Eager promotion No quick demotion	★★★	★
	No promotion No quick demotion	★	★★★★★
	Lazy promotion Quick demotion	★★★★★	★★★★★

How does SIEVE work?

For SIEVE, we denote visited bits as 1 (orange in the following illustration) and not visited bits as 0 (white in the following illustration). Additionally, we have a SIEVE hand, pointing to a bit to check. Suppose now we've known that "D" is a cached hit, the SIEVE hand is pointing as "K."

cache hit on D



If we've encountered a cache miss and want to insert a new bit, we first reset the current visited bit, "K" here, and move the SIEVE hand to the previous unvisited bit. Then we evict this unvisited bit, which can be considered as unpopular so we can evict it, and finally insert a new bit at the start of the queue.

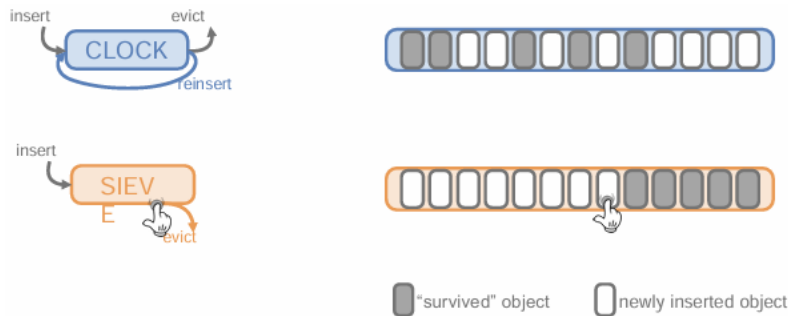
cache miss



In this case, if we want to insert another new bit and want to remove the previous bit we've inserted, it becomes very easy as the previous inserted bit is just before "D."

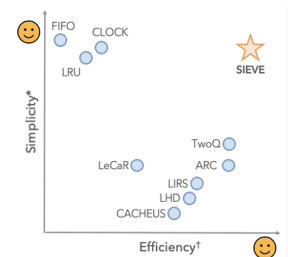


Because of these properties, SIEVE is also very efficient to separate old and new objects (bits), illustrated as follows.



SIEVE takeaway

- Lazy promotion and quick demotion are key to efficient eviction algorithm
- SIEVE uses a moving hand to (1) retain popular objects in place, and (2) remove unpopular objects quickly
- The simplest algorithm with state-of-the-art efficiency and scalability

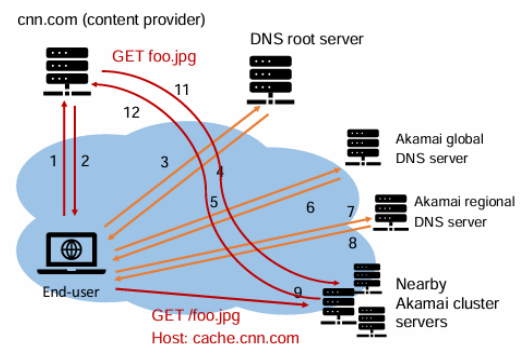


6. Load balancing

- Global load balancing: DNS-based (Akamai) and anycast-based (Cloudflare)
- Selecting server inside cluster: Consistent hashing

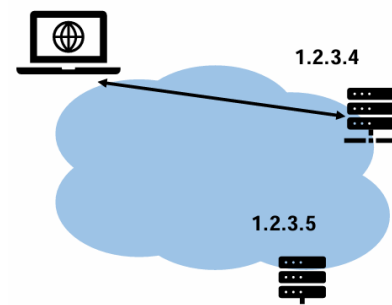
Global load balancing DNS

- Advantages:
 - Avoid TCP setup delay
 - DNS caching reduces overhead
 - Relatively fine control
- Disadvantages
 - Based on IP address of local DNS server
 - “Hidden load” effect
 - DNS TTL limits adaptation



Global load balancing - anycast

- Advantages
 - No extra round trips
 - Route to nearby server

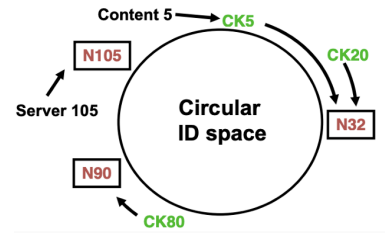


- Disadvantages
 - Does not consider network or server load
 - Different packets may go to different servers
 - Used only for simple request-response apps

Selecting server inside cluster

“Consistent hashing” algorithm:

- $\text{content_key} = \text{hash}(\text{URL}) \bmod N$
- $\text{node_key} = \text{hash}(\text{server_id}) \bmod N$
- Rule: place content on server whose node_key is the successor server of URL's content_key
- One of the main goals of consistent hashing is to reduce data redistribution



IV. Related readings

1. (Optional) [FIFO can be Better than LRU: the Power of Lazy Promotion and Quick Demotion](#)
2. (Optional) [FIFO Queues are all You Need for Cache Eviction](#)
3. (Optional) [Segcache: a memory-efficient and scalable in-memory key-value cache for small objects](#)