CS 7172
Parallel and Distributed Computation

Distributed Cache

Kun Suo

Computer Science, Kennesaw State University

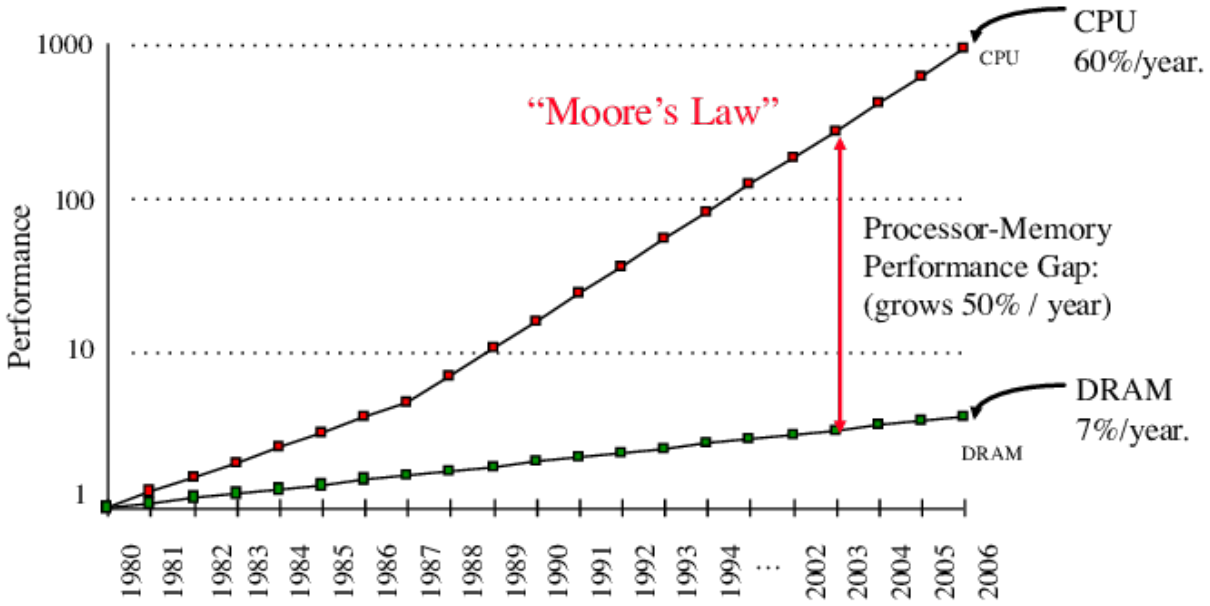
https://kevinsuo.github.io/

Outline

- Computer networks, primarily from an application perspective
- Protocol layering
- Client-server architecture
- End-to-end principle
- TCP
- Socket programming

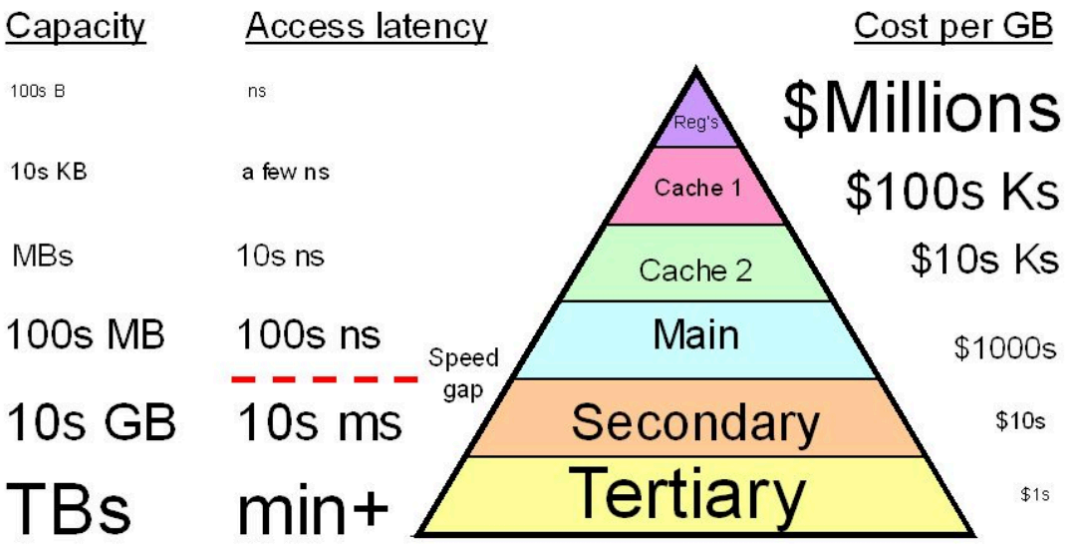
Why Cache?

- From the perspective of the architecture of a single computer, memory and processor speeds are different. If cache technology is not used, the performance of the processor will be greatly limited.



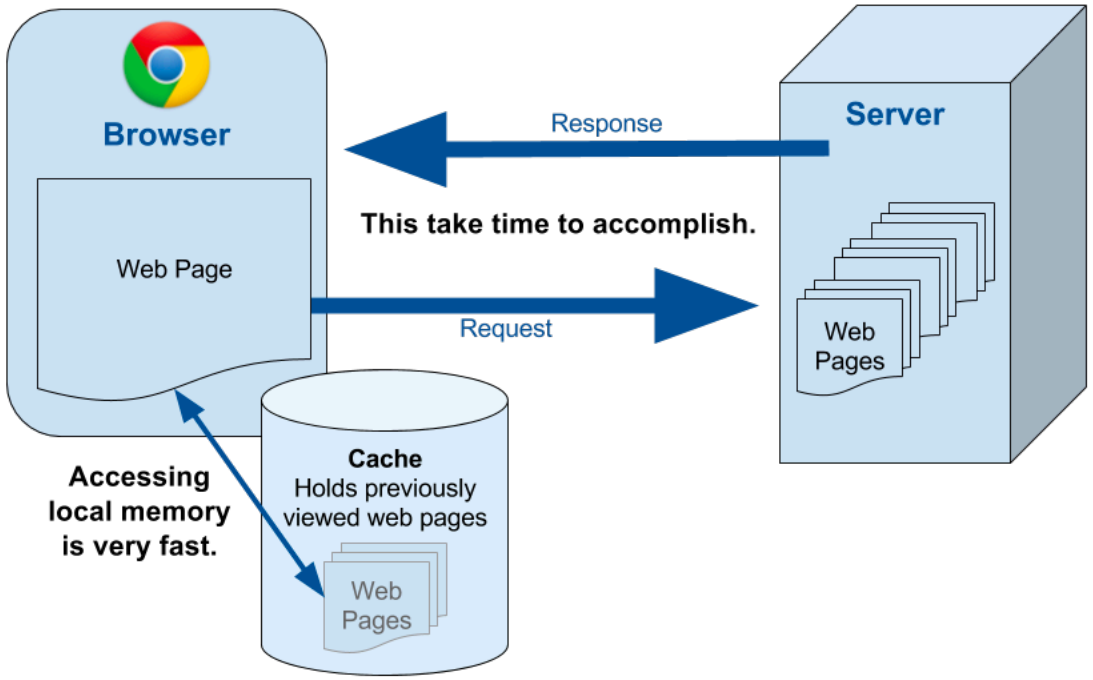
Why Cache?

- The application must interact with the database, and the data in the database is stored on the disk. Each request must interact with the disk, and the performance of disk access is very low, which causing significant access delay



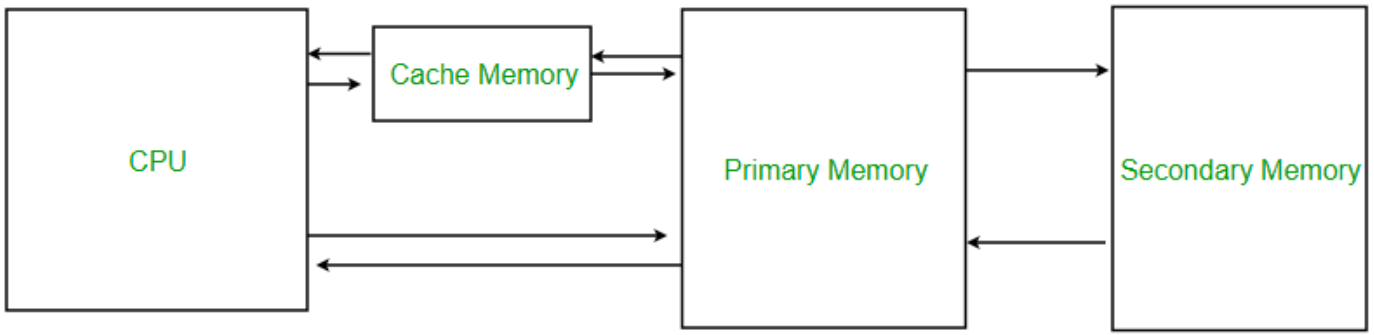
Why Cache?

- Network access: if there is no caching on the browser, you must interact with the remote machine each time you visit the host, which is very slow



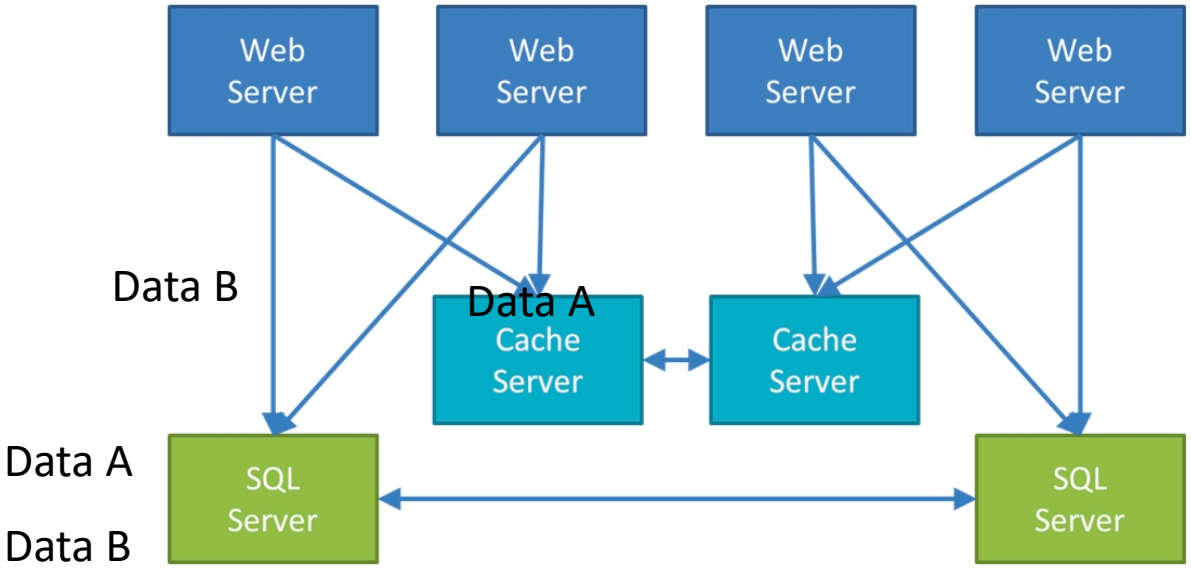
Cache and Distributed Cache

- Cache: use a faster storage device to store some frequently used data for quick access by users. Users don't need to interact with slow devices every time, so it can improve access efficiency



Cache and Distributed Cache

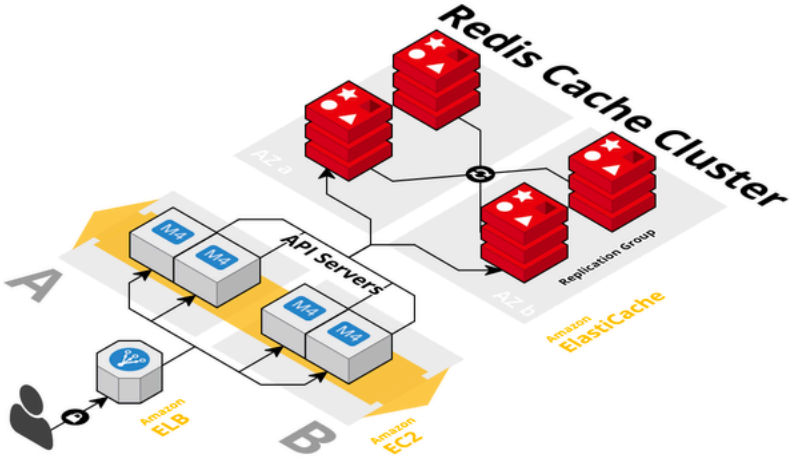
- Distributed cache: in a distributed environment or system, store some popular data near the user and the application, and store the data on faster devices to reduce the delay of remote data transmission



Distributed Cache in Redis

- Redis: Remote Dictionary Server

- Store data in memory with a dictionary structure.
  Applications can read and write data stored in Redis directly in memory



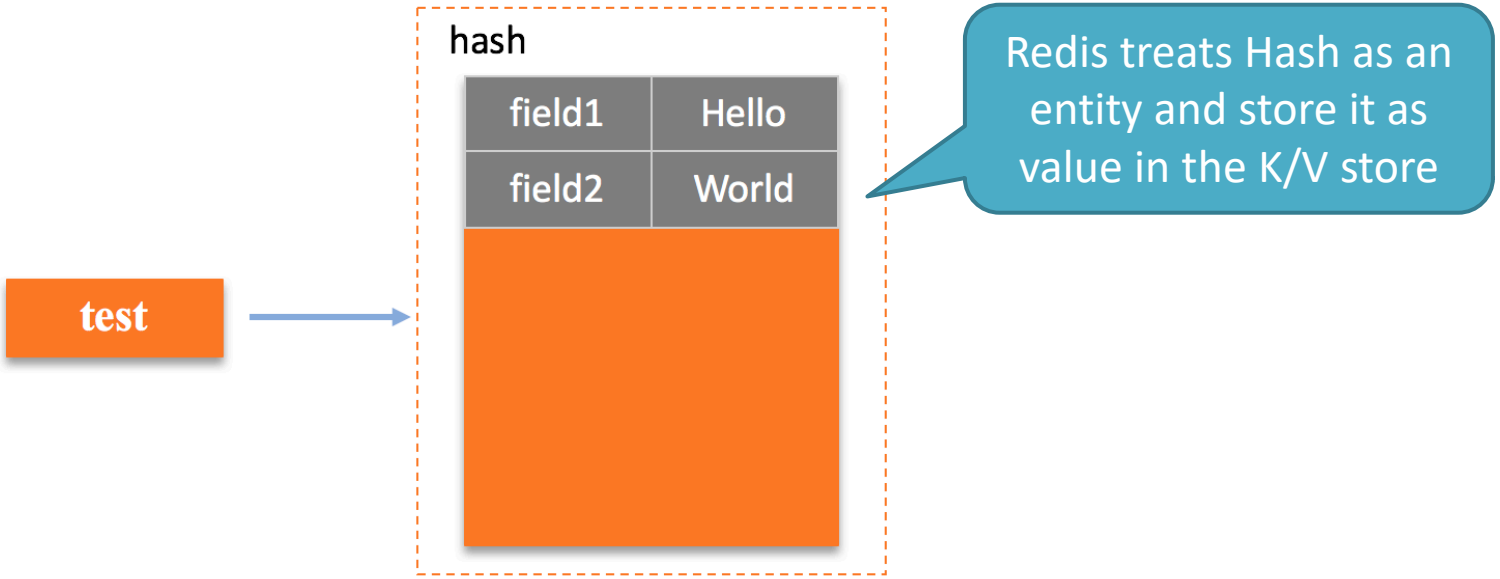
- Decentralized structure, each node is responsible for storing some data

Distributed Cache in Redis

- Three features that are most closely related to the cache in Redis:
  - support for multiple data structures
  - support for persistence
  - active-standby synchronization

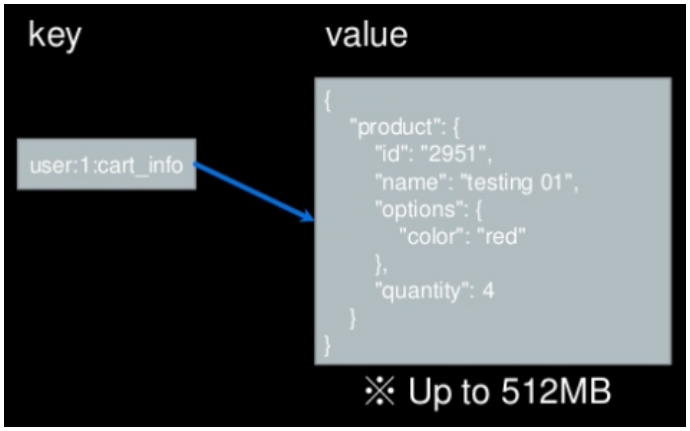
Support for multiple data structures

- Redis is a memory-based key-value database. The supported data structures include not only simple k/v types, but also complex types such as List, Set, and Hash.

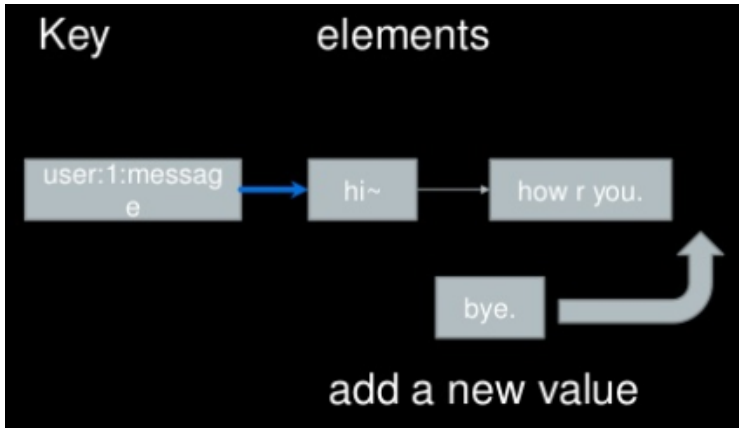


Support for multiple data structures

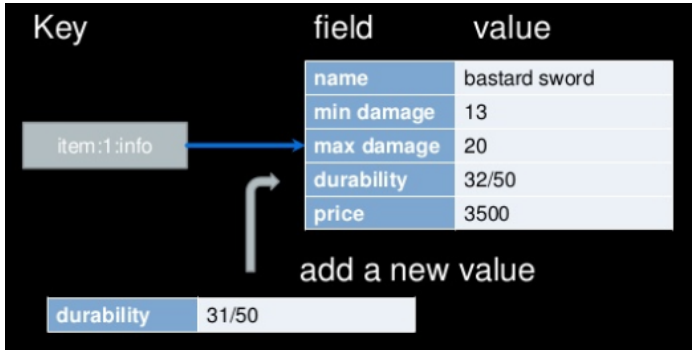
String



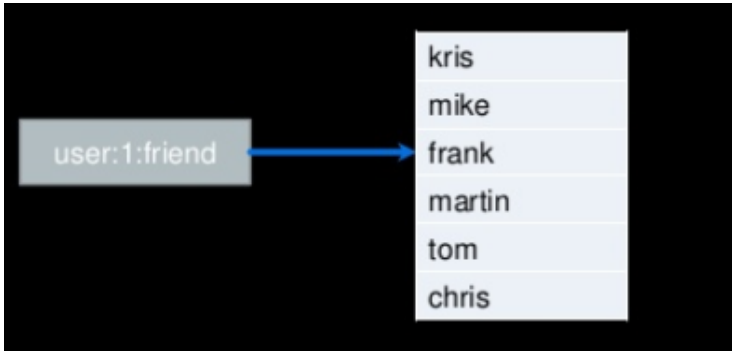
List



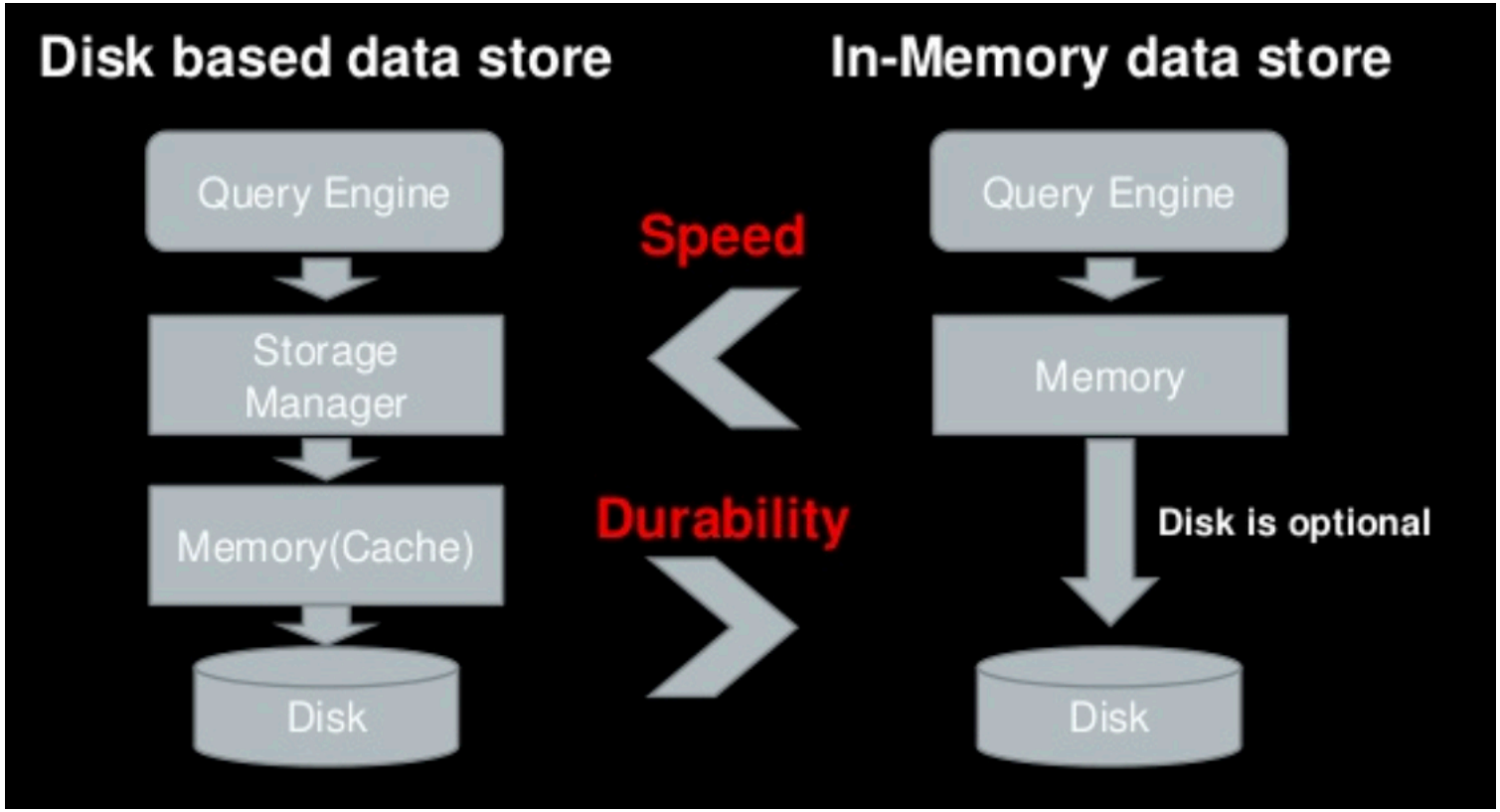
Hash



Set



Support for persistence



Support for persistence

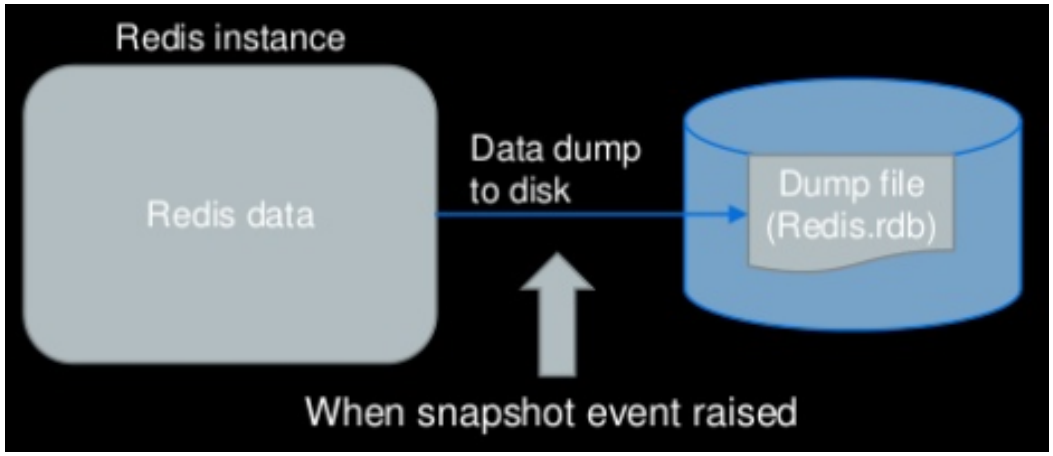
- Although the data stored in Redis is based on memory, it provides two ways for persistence: RDB and AOF.

- RDB (Redis DataBase), also known as the snapshot method: Redis periodically backs up the data in memory to disk to form a snapshot.

- AOF (Append Only File): records all update operations in Redis

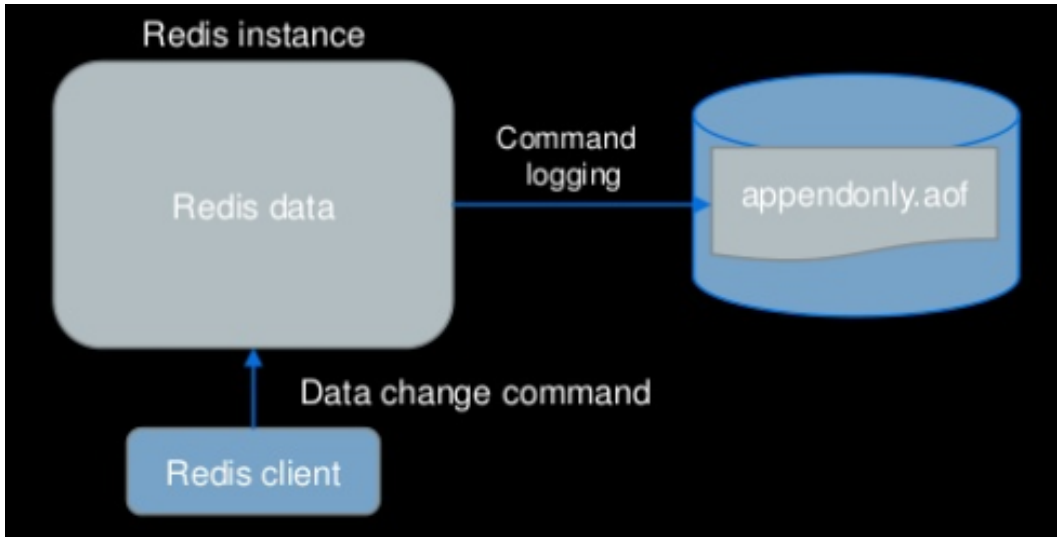
Support for persistence

- RDB (Redis DataBase), also known as the snapshot method: Redis periodically backs up the data in memory to disk to form a snapshot.



Support for persistence

- AOF (Append Only File): records all update operations in Redis

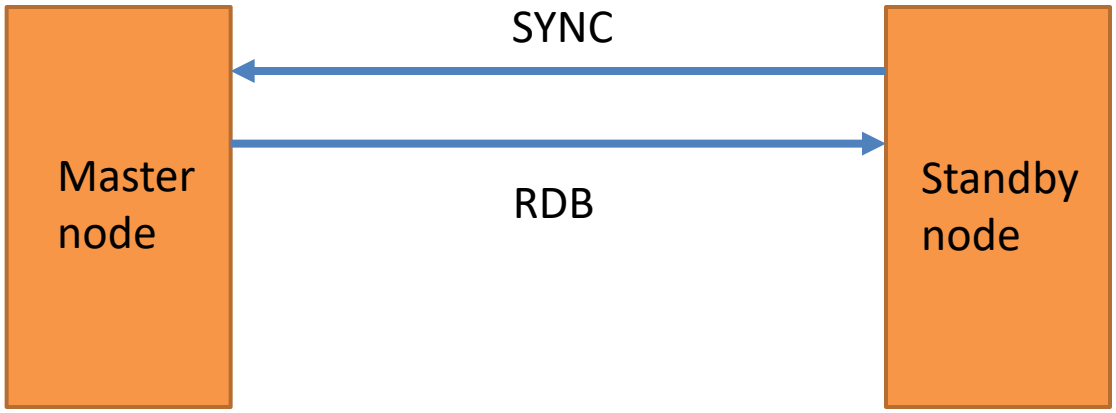


Active-standby synchronization

- The complete synchronization:

  1. When the standby server starts, it will send a SYNC command to the master server;

  2. After receiving the command, the master server will generate an RDB (snapshot) file and record the new write operations performed from now on;

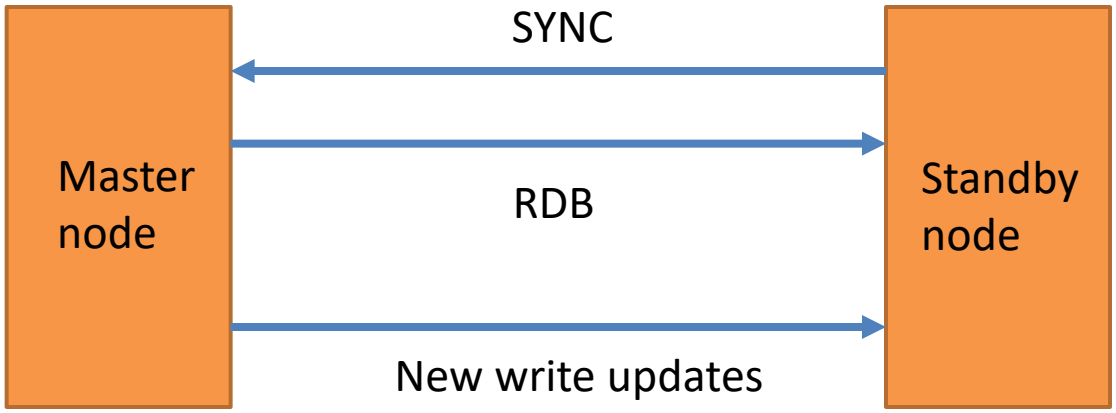


Active-standby synchronization

- The complete synchronization:

  3. When the RDB is created, it will send it to the standby server, and the standby server will update the data through the RDB file.

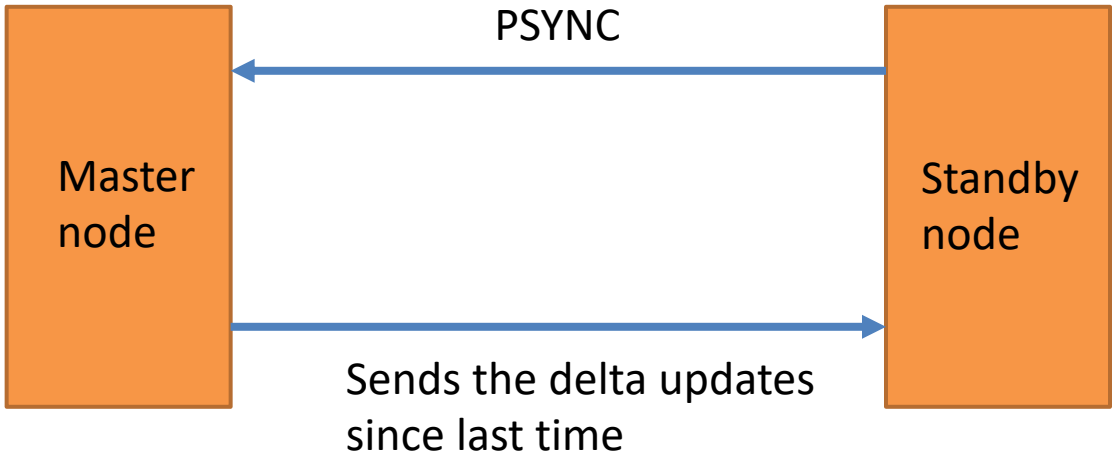
  4. After the update is complete, the master server sends the write operation of the new records to the standby server. Then, the data is consistent.



Active-standby synchronization

- The part synchronization:

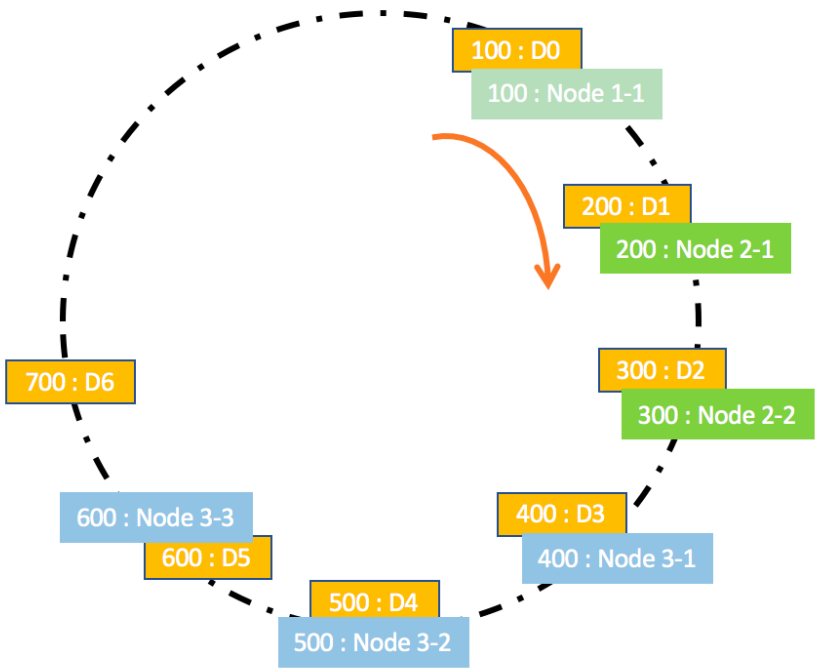
  Master node only sends the updates to the standby servers and the standby servers perform the writes and keep data consistent



Distributed Cache in Memcached

- Memcached is a high-performance memory-based key-value cache database

- Memcached adopts consistent hashing with virtual nodes



Support for multiple data structures?

- Memcached only supports simple k / v data types.

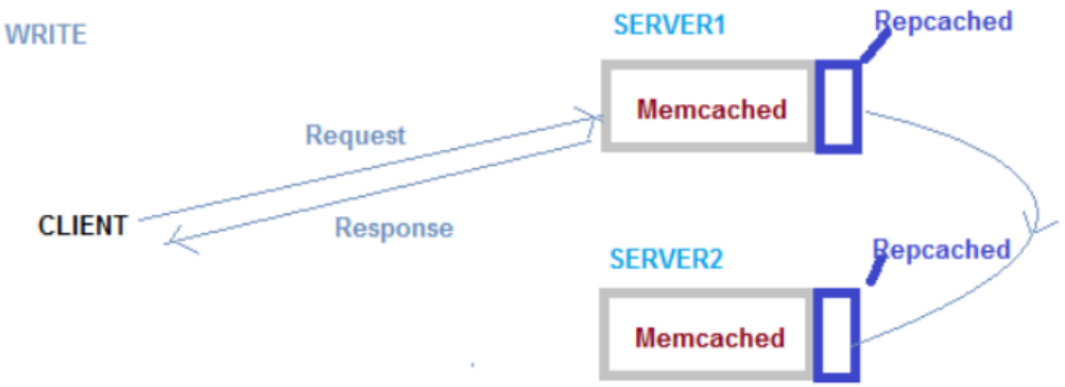
- If users want to store complex data types, such as List, Set, and Hash, they need to handle it by themselves to convert it to a string, and then store it.

Support for persistence?

- Memcached does not support persistence
- All data in Memcached is located inside memory

Active-standby synchronization?

- In Memcached, there is no communication between the servers
- It does not support master/slave by itself. If users want to achieve high availability, they need to implement it through a third party application, such as Repcached



Comparison between Redis and Memcached

	Redis	Memcached
Storage	Memory + Disk	Memory
Distributed cluster	Redis cluster, each node is responsible for part of hash	Consistent hashing with virtual nodes
Support for data structure	Besides K/V, it supports list, set, hash, etc.	Only K/V
Persistence	Support	Not support
Active-standby synchronization	Complete or part synchronization	Not support, needs 3rd party application

Data replication

- Data replication is a way to implement data backup
- With data replication, when a node fails, data can be obtained from other nodes that store the data to avoid data loss, thereby improving the reliability of the system.
- How to keep the data in the primary and secondary nodes consistent is the main challenges.

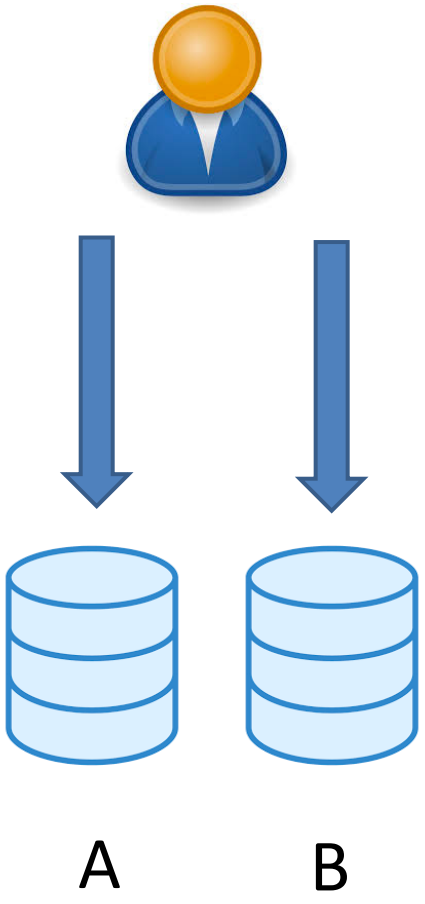
Synchronous replication

- When a user requests to update data, the primary database must be synchronized to the standby database before it returns to the user. If the primary database is not synchronized to the standby database, the user's update operation will always be blocked.

- It guarantees strong data consistency, but sacrifices system availability

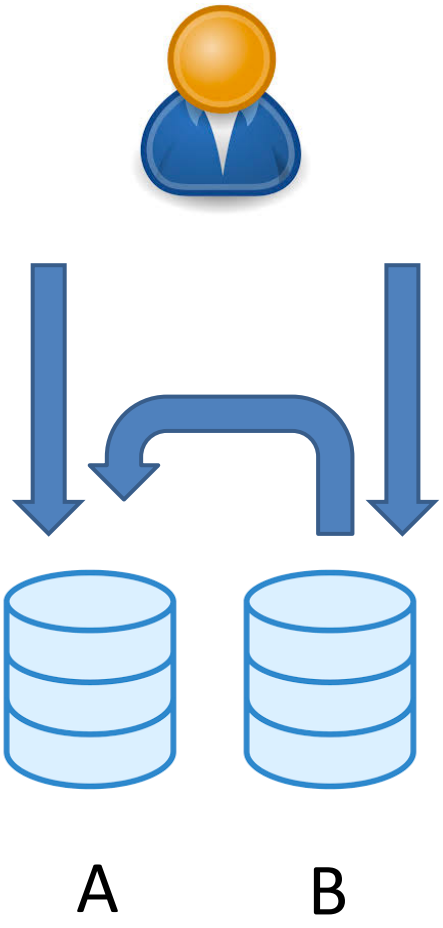
Synchronous replication example

- Two nodes database: A and B
- A is the primary database and B is the standby database
- **Read**: both nodes can receive the user's read request, and then return the data of this node to the user in time



Synchronous replication example

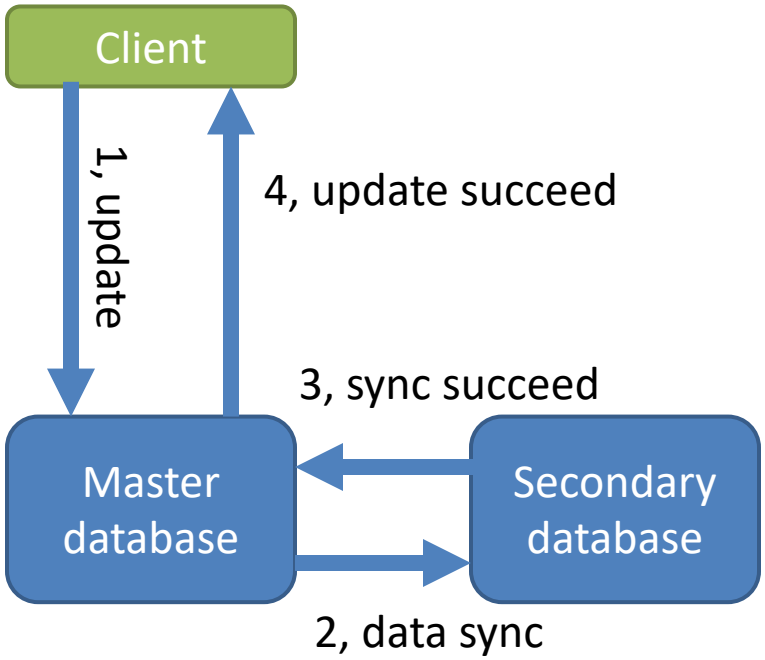
- **Write**: if the user sends a write request, the write operation must be performed by the master node. Even if the user sends the write request to the standby node, the standby node will forward the request to the master node.



- Read and write separation system
  - E.g., MySQL

Synchronous replication example

1. The client sends an update operation V to the primary database and sets X to 2.
2. The primary database will synchronize the write request to the standby database.
3. After the standby database operation is completed, the primary database will be notified of the synchronization success.
4. The primary database will tell the client that the update operation was successful.



Synchronous replication

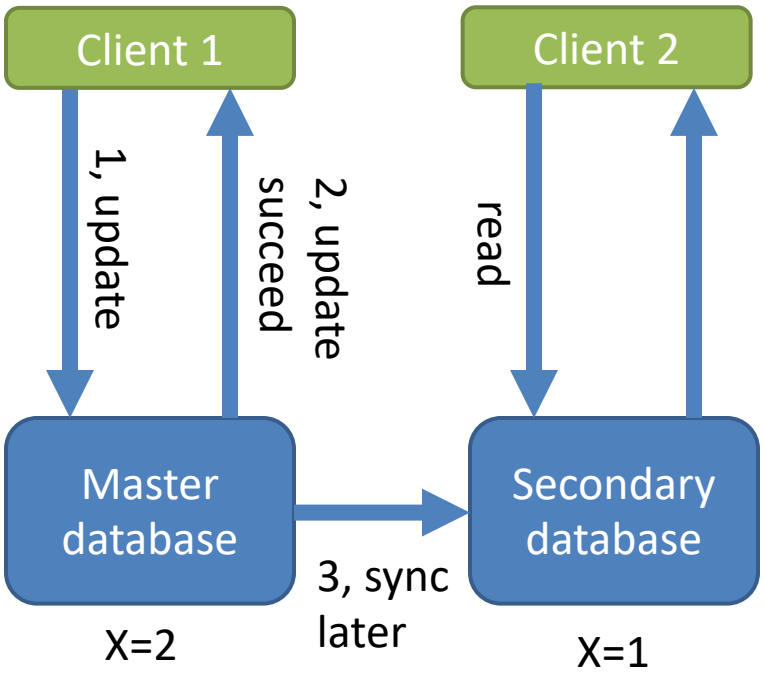
- Scenario:
  - It is often used in distributed database with active/standby scenarios or where strict data consistency are required, such as financial and transaction scenarios.

Asynchronous replication

- When a user requests to update data, the primary database can directly respond to the user after processing the request, without waiting for the standby database to complete synchronization
- The user's update operation will not be blocked even the standby database has not completed data synchronization
- This method guarantees the availability of the system but sacrifices the consistency of the data.

Asynchronous replication example

1. Client 1 sends update operations to master database and set X as 2
2. The database sends the updated results back directly
3. The synchronous operation will be performed later
4. When client 2 read X from standby database before the synchronization, it only gets the old results



Asynchronous replication

- Scenario:
  - Applications have high latency requirements in response to user requests, e.g., online shopping websites
- Example:
  - Redis cluster

Semi-synchronous replication

- After the user sends a write request, the primary database will perform a write operation and send a synchronization request to the standby database, but the primary database can respond to the user without waiting for all the standby databases to respond to the successful data synchronization, which means that the primary database can wait for ***a portion of the standby database*** to complete synchronization

- Example: Zookeeper cluster

Oracle cluster provides three ways for data replication

- **Maximum protection mode:** for the write request, the primary database is required to complete data synchronization of at least one standby database before it can be successfully returned to the client.
- **Maximum performance mode:** For write requests, as long as the main database is successfully executed, it can be returned to the client.
- **Maximum availability mode:** This mode means that the system adopts the maximum protection mode under normal circumstances, but when there is a network failure between the master and the backup, it switches to the maximum performance mode. After the network is restored, the standby database performs data synchronization.

Comparison

	Consistency	Availability	Scenario	Example
Synchronous replication	Strong	Weak	Used for strict consistent data scenario such as financial systems	MySQL
Asynchronous replication	weak	Strong	Used for high performance scenarios such as online website	Redis, Oracle
Semi-synchronous replication	Relatively strong	Relatively weak	Used for most of the distributed scenarios	Zookeeper, Oracle, Etcd