A comparison between Redis and Memcached

Chunxiao Zhang Netid: cz31

1. Why are we using Redis and Memcached?

Thinking of one simple scenario, in a user profile management system, which type of operation is the major database operation? Write or read? After the user is signed up, the operation of the modifying profile will be much less than logging in. So, in this system, the requirement of reading is much larger than the one of writing. Considering this feature of this type of system, we may use a cache database to store the user profile information to save the time of query and calculation, so the speed of query can be accelerated and the database load is decreased. The cache may not be barely stored in RAM, it can also be stored in a file system or even CPU. Figure 1 shows how the in-memory database work to reduce database load.

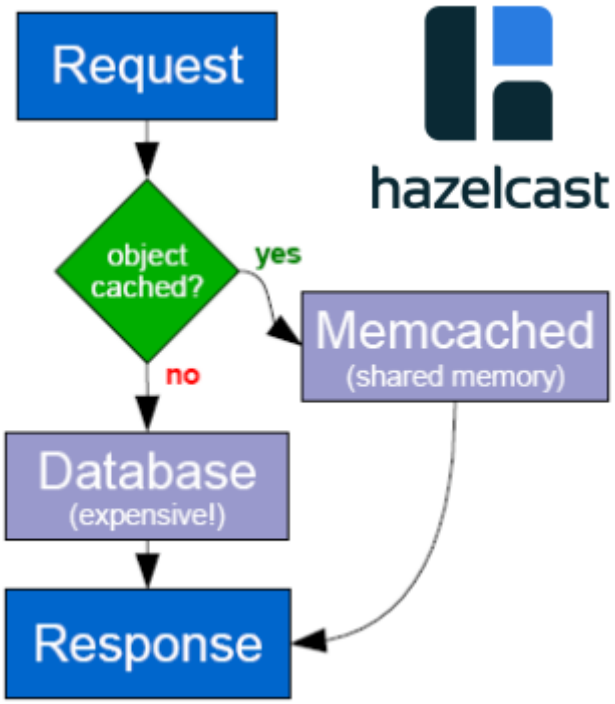

Figure 1 How an in-memory database work

In a large scale system, when performance needs to be improved, caching is always the first step being taken, and Memcached or Redis are typically the first comes to be considered.

2. What is Memcached

Memcached is a high-performance distributed in-memory caching system. Memcached intrinsically is a volatile memory key-value database. It is often used to speed up dynamic database-driven websites by caching small chunks of data (string or objects) in RAM to reduce the times of an external data source (such as a database or API) must be read.

Memcached is a cache-aside database, the application communicates with database and cache separately. As shown in Figure 2, the application will ask Memcached the value of "key"

first, if there is no result in Memcached, the application will ask the database. When the application gets the value from the database, it will store this value in Memcached. Then next time, if the application needs the same value, this value can be retrieved from the Memcached directly. This is a client-server model between the application and the Memcached server, the application needs to know all the existing servers and manage the data allocation among these servers by itself. There is no direct communication between database and cache.

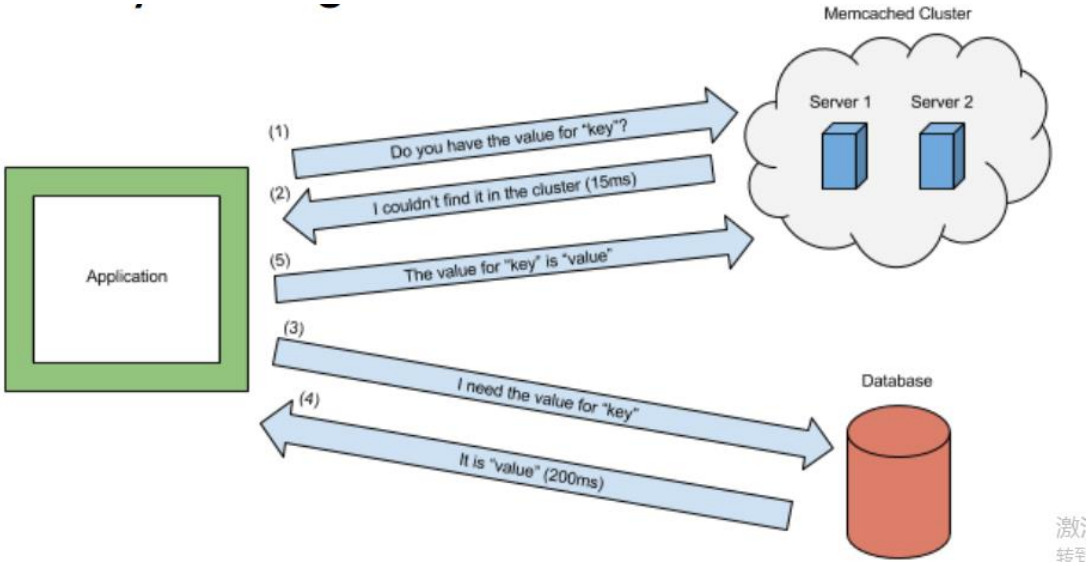

Figure 2 Communication between application, Memcached, and database

3. What is Redis

Redis is a data structure server. It is networked, in-memory, and stores keys with optional durability. The name "Redis" means REmote DIctionary Server. Redis provides key-value cache and storage. Seven types of data structures can be stored in Redis, that is strings, hashes, lists, sets, sorted sets, bitmaps, and hyperloglogs. In order to provide high performance, Redis works with an in-memory dataset. Depending on your use case, you can either keep data persistence or not. Redis also supports trivial-to-setup master-slave asynchronous replication, with very fast non-blocking first synchronization, auto-reconnection with partial resynchronization on a netsplit. Figure 2 shows the functionalities of Redis.

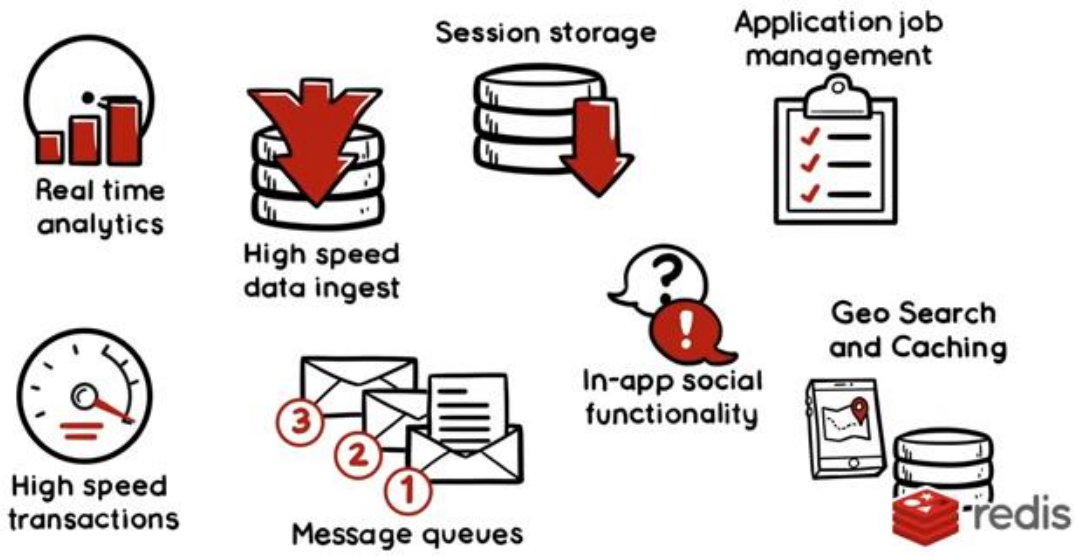

Figure 3 What can Redis do

As shown in Figure 4, Redis is a cache through the database, the backend server will not know the interior structure of Redis, it will not know how the cache and database communicate. When there is a query request, Redis itself will send the data from the cache to the web server, and to make these data persistence, Redis will store these data in the database at an appropriate time.

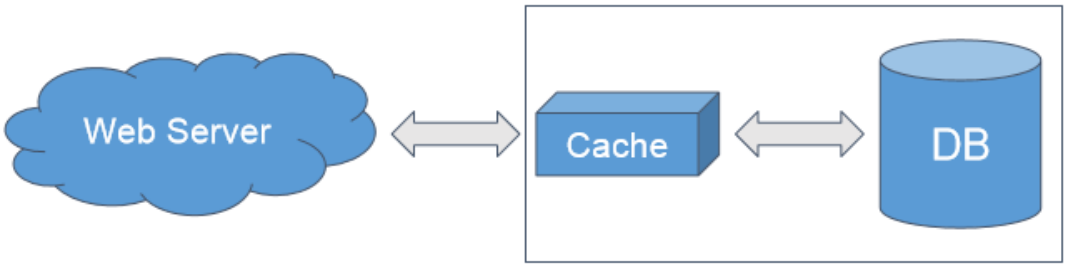

Figure 4 Cache through communication

4. Commons and differences:

Memcached or Redis is a question that always comes up in a database driven Web application design.

In a general way, the basic functionalities of Redis and Memcached are the same. They are all NoSQL in-memory database, both of them can provide high-performance in-memory database service and they can store key-value mapping in memory which makes them useful as a caching layer.

4.1 Value storage

Memcached is a volatile in-memory key-value cache, it can only store string or objects. Redis can not only do the same job as Memcached but also can work as a data structure server. It can store more complicated values such as hashes, lists, sets, etc. Figure 5 shows the data types that Redis can store.

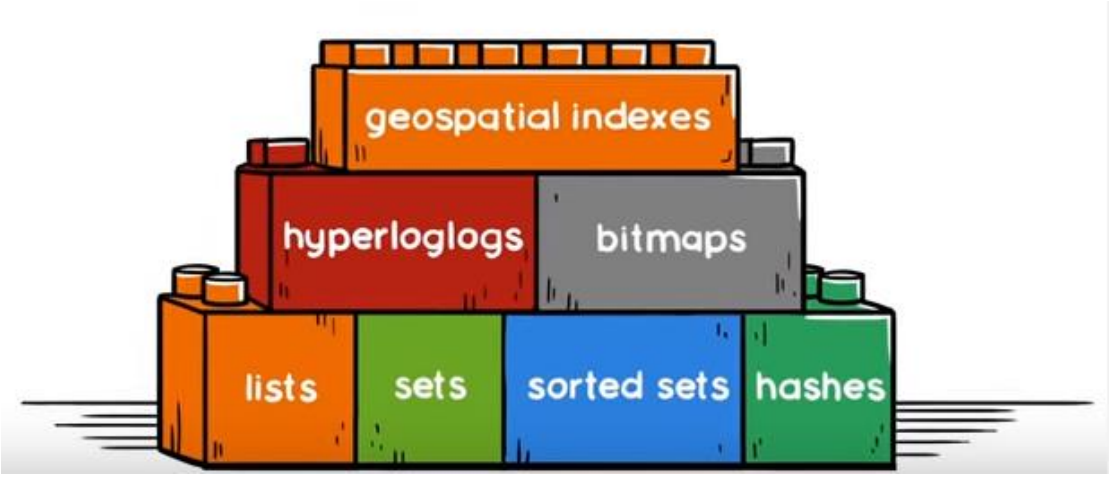

Figure 5 Data types that Redis can store

4.2 Persistence

Memcached can only store data in RAM, in this case, data stored in Memcached cannot be retrieved all the time, all the information will be lost when there is a power breakout.

In Redis, depending on your use case, you can persist it either by dumping the dataset to disk every once in a while (snapshot) or by appending each command to a log which is called append-only file (AOF). The snapshot can save the data in a period to a file and AOF records the modification operations on data. Persistence can be optionally disabled to optimize performance if you just need a feature-rich, networked, in-memory cache.

4.3 Distribution storage

For cache database system, the physical memory size determines the maximum data storage size in this system. (Redis can extend this limitation since it can store data in disk) To extend the storage capability of the database, we need to build a distributed system.

Memcached itself does not support distributed system. The client needs to use a consistent hash to realize Memcached distributed storage. Before the application sends the data to Memcached cluster, it will use the distributed algorithm to calculate the destination node and send data to that node. When the application is querying data, it also needs to calculate the node first, and send a request to that Memcached node.

While Memcached realizes distributed system on the client side, Redis can build the distributed system storage on the server side. Redis allows single node breaking down. In a Redis cluster, as shown in Figure 6, all master nodes are connected to each other, the client is connected to the master nodes directly, there is no proxy layer or central node concept in this system design. Redis divides the key range into 4096 slots, each node can store one or more slots. The client can use an algorithm to find which node stores the data.

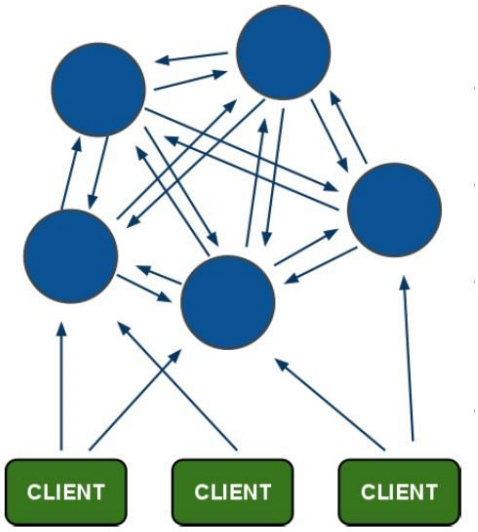

Figure 2 Redis distributed system

To make the database reliable although there is one node breakdown, Redis uses a master-slave model to manage its servers. Each master will have two slave nodes, Redis replicates the data from master node to slave nodes. Replication can be used for implementing a highly available cache setup that can withstand failures and provide uninterrupted service to the application. This is a big advantage in user experience and application performance comparing with Memcached.

Conclusion:

Memcached showed up in 2003 and Redis first came up in 2009. Both of them are popular in-memory cache databases. Actually, in these years, Redis is more popular than Memcached. This tendency can be shown in Figure 7.

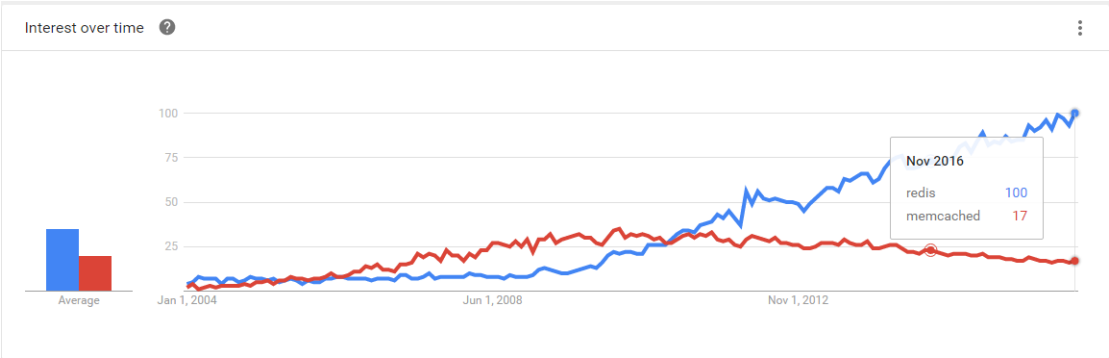

Figure 7 Statistic comparison between Redis and Memcached, from Google trends

Though there is a tendency that more and more people are interested in Redis and Memcached seems to be a little bit out of time, there are still some big companies such as Flicker and Wikipedia are using Memcached. There are no obvious differences in processing performance between these two databases.

The basic concept of Memcached is simpler than Redis and theoretically, the processing speed of Memcached is superior to the one of Redis, since all of the data in Memcached are stored in RAM. But the data stored in Memcached is not reliable enough in some special scenarios. For example, if the user wants to modify the data, he modifies the cache data first.

But the database breakdown before he changes the database, the data he gets from cache next time will be different from the database's data, this will cause a problem. The following pseudo-code shows this process.

```
cache.set(key, user)
//database breakdown at this time
database.set(user) //this will not work
//This will cause the data from database and cache are different, data you get from cache next time will be some dirty data.
```

If the user focuses more on the persistence and the data synchronous, Redis should be a better choice. Redis ensures the data will not be lost when the users restart their machine or a power breakout happens. Also, the data stored in Redis is more reliable since it will synchronize the data stored in the cache and database.

Overall, if the users have already made many investments on Memcached, they don't have to switch their application to Redis, both of them are mature and reliable, their performance and functionalities are almost the same. But if the users have just started developing Redis, this paper may suggest them to use Redis, because comparing with Memcached, Redis is newer and more versatile.

6. Reference:

[1] lec24_scaling.pdf, Scott Pollack, 11/27/16, https://piazza-resources.s3.amazonaws.com/is3j293rbvv6wv/ivjwwtaesey1hb/lec24_scaling.pdf?AWSAccessKeyId=AKIAIEDNRLJ4AZKBW6HA&Expires=1480279803&Signature=RrCHec5aM3ThILRbZr65PNeqvZo%3D

[2] Introduction to Redis, 11/27/16, https://redis.io/topics/introduction

[3] Redis vs Memcached, K Hong, 11/27/16, http://www.bogotobogo.com/DevOps/Redis/Redis_vs_Memcached.php

[4] Google Trends, Compare redis & Memcached, 11/27/16, Google, https://www.google.com/trends/explore?date=all&q=redis,memcached&hl=en-US

[5] Memcached Main Page, 11/27/16, https://memcached.org/

[6] Comparison between Memcached and Redis, pi9nc, 11/27/16, http://blog.csdn.net/pi9nc/article/details/21001045