Ho Chi Minh University of Science
Information Technology (HQ Program)

INTRODUCTION TO BIG DATA

REDIS

27/04/2022

Class: 19KHMT1 – **Group:** KKAP

PROGRESS: 100%

LECTURERS

Lê Ngọc Thành
Nguyễn Ngọc Thảo
Phạm Trọng Nghĩa

Work Assignments

Name	ID	Tasks	Progress
Ngô Văn Anh Kiệt	**19127191**	Implemented ClickAnalyzer, Connected back-end to Spark and Redis, Report (Demonstration)	100%
Ngô Huy Anh	19127095	Gathered research material, Report (Real-life application), Implemented Django server	100%
Phùng Anh Khoa	19127449	Report (Introduction, Redis Replication), Presentation contents	100%
Triệu Nguyên Phát	19127505	Report (Insight, layout design), Slide animations, Proofread & Quality checking	100%

Table of Contents

1. Introduction

Playing a big role in the 4.0 technology era, Big Data comes with new technologies and storage methods to help solving larger real-world problems. NoSQL is a flexible storage method that can be used to replace highly rigorous SQL. Among them, Redis stands out for using primary-replica architecture, enables highly available solutions that ensure consistent and reliable performance.

In 2009, Salvatore Sanfilippo, the creator of Redis, was looking for a method to store important information on RAM. Being an open source database that has grown rapidly in recent years, Redis assists in the resolution of time-sensitive problems such as caching, sessions, management, leaderboards, real-time systems, and much more... Lately, Redis supports a diverse set of data types and programming languages, making data manipulation and querying simpler and faster.

2. Insight

2.1. Supported Data Structures

Redis, as opposed to Relational Database Management Systems such as MySQL, PostgreSQL or Oracle etc, is primarily a key-value storage. While keys are binary strings, the advantage of Redis is that the value is not restricted to a single data type. Redis has a variety of data structures that allow the storage of the complex objects as well as a rich set of operations on them.

In term of data types, Redis merged number (integer or float) into String, more typical structures can be seen are List, Set, Hash and, featured more special structures, such as

- **ZSet:** known as Sorted Set, where every member of a Sorted Set is associated with a score, that is used to keep the Sorted Set in an ascending order.
- **Bitmaps and HyperLogLogs:** these are actually data types based on the String base type, but having their own semantics.

However, one that originally made the name of Redis is **Streams**, known as a data structure that acts like an append-only log. It enables collecting, persisting, and distributing data at a high speed with sub-millisecond latency. Offer an asynchronous communication between producers and consumers, with additional features such as persistence, look-back queries, and scale-out options similar to Apache Kafka.

2.2. Redis Replication

Replication enables a write operation to a Redis instance (usually referred to as the master) to ensure that one or more instances (usually referred to as the slaves) become exact copies of the master. This method is how Redis supports consistency and failover handling.

Redis Replication works as follow:

- Masters have the version ID, each master takes an offset that increments each byte whenever a replication stream is sent to the replicas, to update the state of the replicas with the new changes modifying the dataset.
- The master starts a background saving process to produce an RDB file. At the same time, it starts to buffer all new write commands received from the clients.
- When the background saving is completed, the master transfers the database file to the replica, which is saved on disk and loaded into memory.
- The master will then send all buffered commands to the replica.

Replicas are widely used for scalability purposes so that all read operations will be handled by replicas and the master only has to handle the write operations. Data redundancy is another popular reason to have multiple replicas.

Replicas are very useful in a master failure scenario because they contain all of the most recent data and can be promoted to master. However, when Redis is running in single-instance mode, there is no automatic failover to promote a slave to master. All replicas and clients connected to the old master must be reconfigured to the new master.

2.3. Architecture

As Redis Sentinel and Redis Cluster are distributed systems, it is fair to analyze them using the **CAP** theorem. In a distributed system, network partitions are unavoidable, so it should ensure either consistency or availability, i.e., Redis should be either **CP** or **AP**.

Theoretically, Redis are neither consistent nor available under network partition[2]. It shows that both Redis Cluster and Redis Sentinel, to distribute Redis, are faulty in that they loose both data consistency and also can loose persistent data in the presence of network partition[3].

To go over the CAP theorem:

Any distributed data store can only provide **two** of the following **three** guarantees:

- **Consistency**: Every read returns the most recent write.
- **Availability**: Any operation receives a response stating whether it has succeeded or failed.
- **Partition tolerance**: The system goes on when a network partition - a network failure that causes the members to split into multiple groups - occurs.

In details, when a partition happens:

- The minor parition becomes unavailable. Even though the majority partition will still be accessible, it certainly lost the **Availability** characteristic.
- The Redis itself uses asynchronous replication. Though the majority partition will still execute every commands when network partition is happening, the minor partition will not, thus broke the **Consistency** characteristic.

However, Redis Cluster tried to improve the **Consistency**, by supporting synchronous writes when absolutely needed, implemented via the 'WAIT' command. When 'WAIT' is configured, the master will first try to send the writes to an also configured number of replicas, or if not, a configured waiting time before responding to the clients.

Nevertheless, Redis Cluster does not implement full consistency even when the synchronous replication method is used. It is always possible, under more complex failure scenarios, that a replica that was not able to receive the write will be elected as master.

Consistency	Availability	Partition tolerance
Weak (default configuration) Strong (with 'WAIT' command)	Partly with Redis Sentinel	Must-have as network partition is unadvoidable in a distributed database

Analyzing Redis in CAP theorem

2.4. Main functions

Thanks to its speed and flexibility, Redis can be used for a variety of purposes. Some are even considered to be more powerful than other database management systems.

- **Caching:** Redis's read and write speeds are extremely useful as cache memory. It can also be used as a temporary database or a location to share data between applications. Further to that, Redis can also be used as a Full Page Cache for the website. And, because of its consistency, users will most likely not notice any latency loading the page even when Redis is being restarted.
- **Counter:** Redis's sets and sorted sets declare its strength the most when the parameter changes at a rapid pace while the data is stored in RAM. Such cases occur pretty frequently in real life, counting the views on a website or ranking the rank in a game are two examples. Redis also provides a thread-safety support, which allows it to synchronize between requests rather easily.
- **Publish/Subcribe:** Redis supports the creation of channels in order to exchange data and information between publishers and subscribers. Similar to Socket Cluster's channels or Apache Kafka's topics, Redis's Pub/Sub can also be used to track connections in social networks or chat systems.

- **Queue:** Redis supports list storage and a wide range of manipulation with list elements, making it suitable for use as a message queue to handle requests in a sequential manner.

2.5. Salient features

The strength of Redis is that it has a very different data model comparing to a traditional RDBMS since the client commands are not executed through queries, but rather specific operations to be performed on the data. As a result, the data must be stored in a way that allows for quick access without the use of a database system, while also aggregating other common features of a traditional RDBMS.

- **High performance:** Having Redis's data stored in main memory, accessing data are guaranteed to have low to extremely low latency with high throughput. Since the disk access is not demanded, read and write transaction latency is decreased to microseconds, enhancing operation speed and enabling for more than million requests to be processed at any given moment.
- **Flexible data structure:** Redis supports a wide range of data structures, allowing it to adapt to any application requirement.
- **Open source:** Not only being simple to use, Redis also has an adequate amount of documentation for research and development. It also supports a wide variety of languages (Java, PHP, Python, C/C++/C#, JavaScript, Node.js, Ruby, R, Go, and so on), making coding less distressing and more structural (instead of having to query traditionally).

Therefore, Redis is evaluated as suitable for:

- Caching (web page cache, query results, etc.)
- Session application management or Session storage
- Real-time analytics (game's leaderboards, weather forecast, etc.)
- Real-time communication (chat app)
- Multimedia content development
- Applications that make use of geospatial data that is updated in real time
- Machine Learning (training AI model in fast speed)

2.6. Real-life applications

As previously stated, the variety of options for Redis usage has made it popular when accessing speed is taken into account. To demonstrate, we'll look at some applications that used Redis as a component.

Pinterest – Social Media

Pinterest used Redis as a cache to store several data:

- A list of users that you are following
- A list of users that are following you

Similarly, Twitter and Instagram also use Redis to store their timelines, feeds, web page caching and social graphs

Valorant – Game

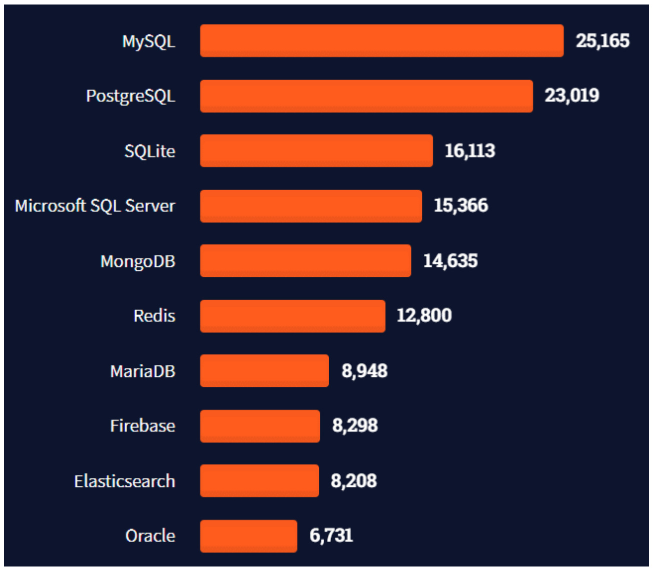
Valorant is an FPS (First-Person-Shooter) video game. There, the developers used Redis's hashes to know whether the user is disconnected or their PC is crashed. Storing the User ID in a Redis cluster based on the modulo of their player ID hash, it counts whether their inactive time exceeds the threshold. Thus it has the load balancing across server instances

Whatsapp, Telegram – Chat App

Whenever a new message is sent or published, it needs the broadcast mechanic to deliver the message to all listeners and/or subcribers. Whatsapp, Telegram used Redis's Pub/Sub model, supporting this feature in a tremendously fast speed.

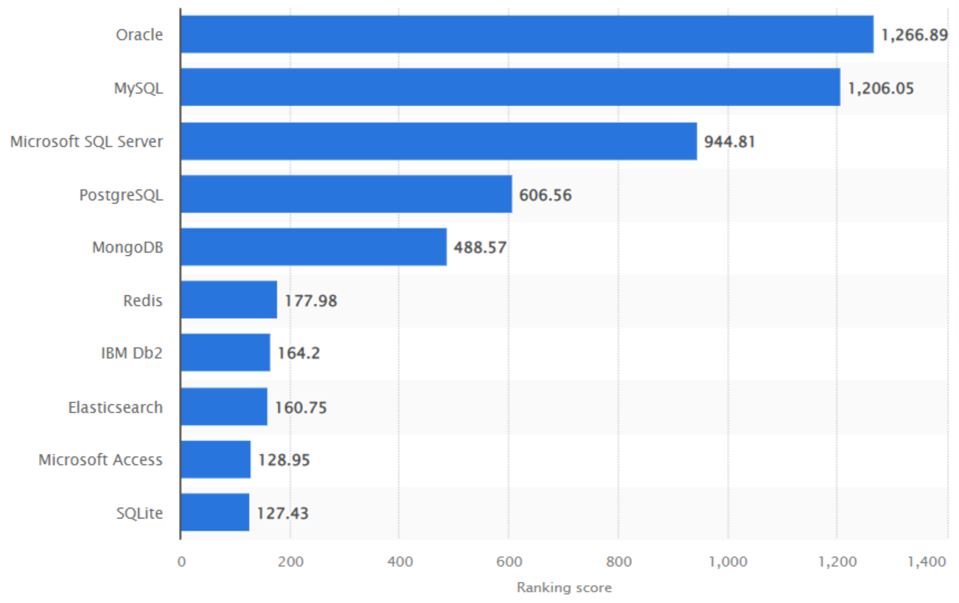
2.7. The popularity

By 2021, Redis was ranked at #6 in DB popularity on StackOverFlow Developer Survey with 12800 votes from the Professional Developers in a total of 53312, winning over Firebase and Oracle.

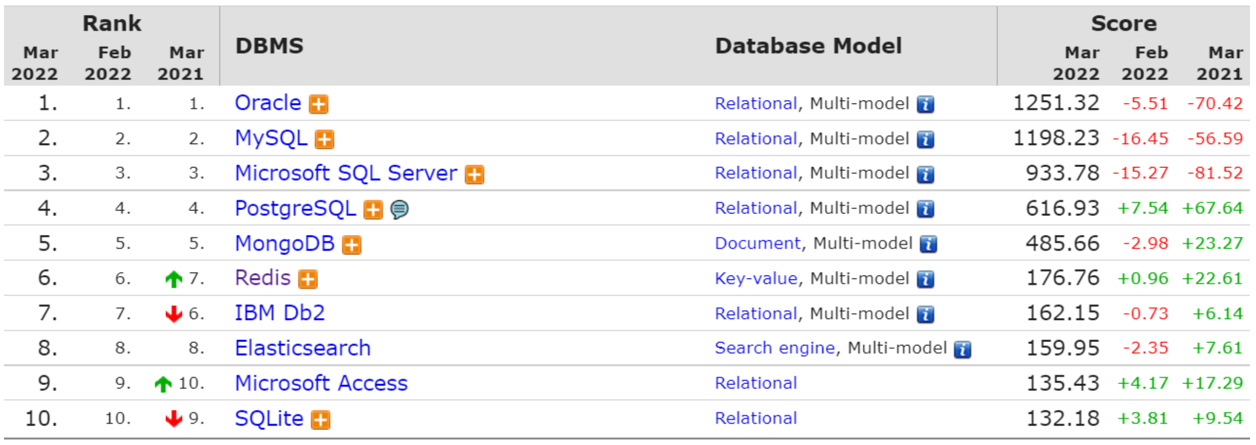


StackOverFlow Developer Survey 2021, Databases category, Profession Developer filter, showing in #Responses (source)

As of January 2022 on Statista, Redis ranked #6 as the most popular database management systems worldwide.



By March 2022, Redis is ranked on #6 in popularity on DB-Engines Ranking in overall category, and #1 in the specific key-value stores category.



Rank			DBMS	Database Model	Score		
Mar 2022	Feb 2022	Mar 2021			Mar 2022	Feb 2022	Mar 2021
1.	1.	1.	Oracle	Relational, Multi-model	1251.32	-5.51	-70.42
2.	2.	2.	MySQL	Relational, Multi-model	1198.23	-16.45	-56.59
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	933.78	-15.27	-81.52
4.	4.	4.	PostgreSQL	Relational, Multi-model	616.93	+7.54	+67.64
5.	5.	5.	MongoDB	Document, Multi-model	485.66	-2.98	+23.27
6.	6.	↑ 7.	Redis	Key-value, Multi-model	176.76	+0.96	+22.61
7.	7.	↓ 6.	IBM Db2	Relational, Multi-model	162.15	-0.73	+6.14
8.	8.	8.	Elasticsearch	Search engine, Multi-model	159.95	-2.35	+7.61
9.	9.	↑ 10.	Microsoft Access	Relational	135.43	+4.17	+17.29
10.	10.	↓ 9.	SQLite	Relational	132.18	+3.81	+9.54

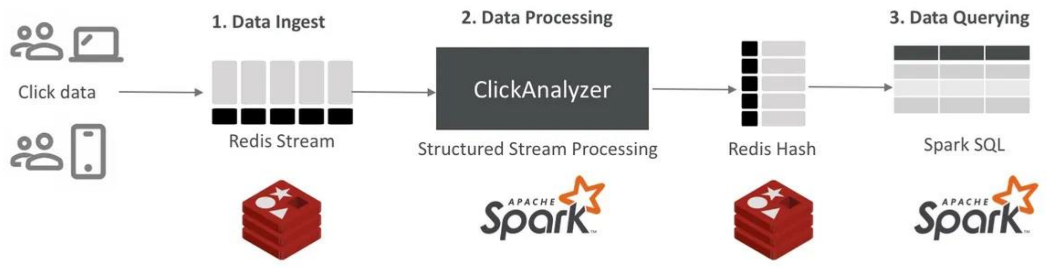
Top 6 on DB-Engines Ranking as of March 2022 (source)

Do note that, Although Oracle wins over Redis in term of popularity, its difficulty made Oracle not be able to win over Redis in Survey.

3. Demonstration

In this demonstration, we will show how Redis can be used as a back-end database for popular data processing frameworks such as Apache Spark. Let's assume the following scenario: an advertisement company wants to analyze which ads gather more clicks so they can show more of those ads to increase profits.

One solution is to set up a data streaming pipeline to record all ad clicks with Redis Stream, then use Apache Spark to process them and compute real-time click counts for each ad, we can store the click counts into the Redis Hash then use Spark SQL module to query from Redis with low latency. Here's an image showcasing our solution architecture:



Spark-Redis click processing architecture (source)

There are 3 main phases in this solution:

- **Data Ingestion**: Everytime a user clicks on an ad, data related to that click (e.g ad ID, ad costs, ...) is sent to a server connected with Redis. The server appends the data to a **Redis stream** using the "XADD" command. This stream acts as a fast buffer that can handle millions of read and write operations per second.
- **Data Processing**: A **ClickAnalyzer** class is implemented through Python with PySpark, Redis-Py and Spark-Redis libraries to make the consumer "digest" the data stream. The analyzer fetches data from the Redis stream and aggregate them into a dataframe holding the ad IDs and the click counts of those ads. From there, we use a Spark writer to write each row that has updates into the **Redis Hash**. All of these can be processed with **Spark Structured Streaming** feature. Although Spark only processes data in micro-batches which causes some slight latency in click count updating, the fast speed of Redis Stream ensures that every click will be recorded in real-time, the analyzer will be able to handle them eventually.
- **Data Querying**: We use **Spark SQL** connected with Redis database to create a **temporary table view** of the Redis Hash, with a column for Ad ID and a column for number of clicks. This phase can run independently from the ClickAnalyzer, and

with how fast Redis is, it only takes a few miliseconds to get the query results. We can then query the table every second to get a live update of the click counts.

This architecture can be applied with almost any front-end and back-end architecture as long as there's a way to connect to Redis and Spark for the Data Ingestion and Querying phases as the Click Analyzer runs independently. In our demonstration, we showed a row of images and their click counts in an HTML webpage, with JavaScript functions to send the click data and request for click counts every second. As for the back-end, a Django web server connected with Redis and Spark session is set up to handle HTTP requests.

One thing to notice from the architecture is that, if the Click Analyzer is slower than the Redis Stream ingestion rate, the speed of our architecture will still be held down by the analyzer, especially when we want to scale out the program. To improve this, or to alleviate the weakness at least, more consumers can be added to digest the stream. As the consistency of Redis allows the creation of a consumer group to digest from that one stream simultaneously, it can eventually match the rate of data ingestion. By setting each consumer's writer to only write the updated rows, there won't be conflicts between them.

4. References

[1] Wikipedia – Redis

[2] Redis Essentials – Online Book – The CAP theorem

[3] Nicolae Marasoiu – Nicolae Marasoiu's answer to What is Redis in the context of the CAP Theorem? on Quora

Other research materials in no particular context:

Redis – Replication

Redis – WAIT numreplicas timeout

Redis – Redis cluster specification

Redis – Redis Cluster consistency guarantees

Vikram Rawat – Vikram Rawat's Answer to Redis availability and CAP theorem on Stackoverflow

Redis – Redis persistence

Redis – HyperLogLog

(Demonstration) Redis – Redis Streams + Apache Spark Structured Streaming