# Database Cashing in-memory with Redis NoSQL Databases

Artan Ajredini
**Contemporary Sciences and Technologies, Software Engineering, South East European University, Tetovo, Macedonia**
**aa28294}@seeu.edu.mk**

## Abstract
Today's data is distributed by nature, the popularity of NoSQL databases using Redis has increased due to the need of processing vast amount of data faster than the relational database management systems (RDMS) by taking the advantage of highly scalable architecture. It examines the need of NoSQL and how they have become an important alternative to relational databases. The importance of Caching is a common technique that aims to improve the performance and scalability of a system.
Caching is a technique used to accelerate application response times and help applications scale by placing frequently needed data very close to the application. Redis, an open source, in-memory, data structure server is frequently used as a distributed shared. In this paper we will present what Redis is and what cashing is methods to use cashing with NoSQL databases in modern applications, Moreover we will demonstrate the usage of object cashing in Redis.

## Keywords
Key-Value Stores, Redis, Relational database, NoSQL.

### 1.INTRODUCTION

In recent period, in-memory data processing is becoming more and more valuable as it is essential to examine a huge amount of information in shorter duration of time Working with large amount of data is always the problem in development and daily maintenance. If SQL is used to store that huge amount of data, it is impossible to process unpredictable and unstructured information.
But social media network demands for an occurrence-oriented database that is very flexible and operates on a schema of less data-model. So SQL can not be fitted for this job. NoSQL, the ability to support large volume of read-write operations and store generic objects, such as JSON, simply contributes to data consistency in distributed system. This shows that NoSQL database is a great choice for processing big and unstructured data. Given the number of data, data access pattern and characteristics of social media, NoSQL has its own indispensable advantages. As a matter of fact, NoSQL has already been widely used in modern software development web-based applications and enterprise solutions. In recent period, in-memory data processing is becoming more and more valuable as it is essential to examine a huge amount of information in shorter duration of time.
NoSQL has many benefits including simplicity of design, scalability of data model, concurrency control, consistency in storage, etc. Redis is an in-memory data structure store, remote database that offers high performance, is a flexible and open source. In this section, we will mainly review and demonstrate one of the most popular NoSQL databases. Redis is in-memory open source is used as cashe but is also a Database. and advanced key-value store database and if you want to store more data you need to have a large of capacity of ram, It may also be persistence so that the data you have on the ram is stored in a file. It is often referred as data structure server. When using an in-memory database like Redis, one of the first questions that's asked is "What happens when my server gets turned off?" Redis has two different forms of persistence available for writing in-memory data to disk in a compact format.

### 2.RELATED WORK

In-memory data caching can be one of the most effective strategies for improving your overall application performance and reducing your database costs.
Caching is a technique used to accelerate application response times and help applications scale by placing frequently needed data very close to the application. In modern computing environments, processes are run at different layers, from low-level processes that are embedded in the hardware to high-level abstractions, such as clusters. Redis, an open source, in-memory, data structure server is frequently used as a distributed shared cache (in addition to being used as a message broker or database) because it enables true statelessness for an applications' processes, while reducing duplication of data or requests to external data sources.
Redis have a wide usage in social media applications working on large amount of data. In present Big data processing, in-memory enumerate has become popular because to increase capacity and high throughput of main memory. When you're building distributed applications that require low latency and scalability, disk-based databases can pose a number of challenges. A few common ones include the following:

.

However, the data retrieval speed from disk plus the added query processing times generally put your query response times in double-digit millisecond speeds, at best. This assumes that you have a steady load and your database is performing optimally.
Cost to scale:
Whether the data is distributed in a disk-based NoSQL database or vertically scaled up in a relational database, scaling for extremely high reads can be costly.
It also can require several database read replicas to match what a single in-memory cache node can deliver in terms of requests per second. The need to simplify data access: Although relational databases provide an excellent means to data model relationships, they aren't optimal for data access.
There are instances where your applications may want to access the data in a particular structure or view to simplify data retrieval and increase application performance.
Before implementing database caching, many architects and engineers spend great effort trying to extract as much performance as they can from their databases. However, there is a limit to the performance that you can achieve with a disk-based database, and it's counterproductive to try to solve a problem with the wrong tools. For example, a large portion of the latency of your database query is dictated by the physics of retrieving data from disk

Solely Redis cannot perform well in a real engineering project, but a combo of Redis and SQL, such as MySQL, would incredibly produce a nice teamwork. As previously mentioned, Redis is fast at IO speed, because of its "caching" feature. MySQL is a popular relational SQL, which has a solid performance and active community. The obvious weakness of Redis is the limited amount of storage. Memory is not a ideal solution for storing a large amount of data. Generally speaking, the most common solution is that MySQL or other SQL database would act as the main data warehouse, meanwhile, Redis can be dedicated to areas that need more flexible and efficient performance.
As to a real development project, usually only a small number of data are unstructured, because most of information of world is kind of in a certain structure. Additionally, cloud-computing is growing rapidly, and the local storage would be less essential in the future development. Maybe only the core program and data are running on the local server. If so, Redis will have a much larger role to play in development.
Both relational and NoSQL databases are in-memory databases that provides different mechanism for data storage and retrieval.

### 2.1 NoSQL Database

According to NoSQL data model, the data stores are grouped into four categories are key-value data stores, document stores, column-family stores and graph databases.
Key-value store: The data are stored as key-value pairs. This data structure is also known as "hash table" where the data are retrieved by keys. Most well-known examples of key-value stores are *Redis*.
Document store: The data are stored in collections that contain key-value pairs which encapsulate key value pairs in JSON (Javascript Object Notation) or JSON like documents
Column family: The data are stored as a set of rows and columns where columns are grouped according to the relationship of data.
Graph databases
The graph databases are category of the NoSQL database, stores data in the form of a graph. In graph databases, the graph is made up of two things: nodes act as the entities or objects and edges act as the relationship between the entities or objects.

## 2.2 Redis

### *2.2.1 Key-Value Database*

Redis is usually known as a key-value database. Doesn't it sound like hashmap, one of the most important data structure in computer science? Redis also has five major data structures, Set, Zset, List, Hash, String, which grant it a promising flexibility. Compared with SQL, even with other NoSQL, this group of data structures makes Redis stand out. Although this cool feature might be the specialty of Redis, it could also be difficult for developers.

Some basic operations with redis:

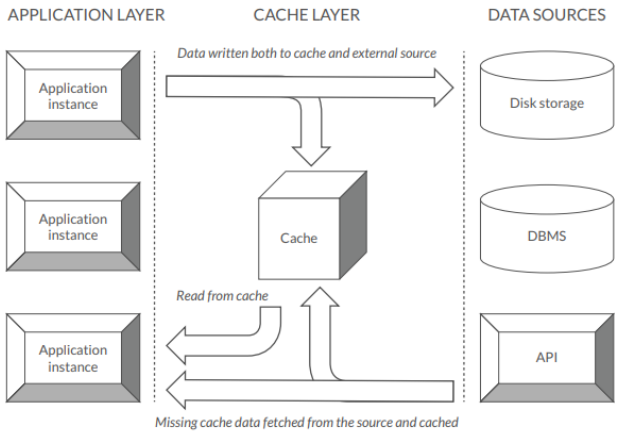
```
SET name "John"     // Set key with string value
GET name            // Get value
SETNX name "Hello"  // Sets key only if not exist

EXISTS name         // Check existence of key
DEL name            // Delete key
EXPIRE name 5       // Delete after 5 seconds

SADD mySet 10 45    // Add elements to set
SMEMBERS mySet      // Get all members of the set
SISMEMBER mySet 10  // Returns whether value exists
SREM mySet 10       // Removes 10 from mySet
```

### *2.2.2 Advantages of Redis comparing to Other NoSQL*

Unlike MongoDB, the unique data storing of Redis makes itself good at IO speed. Its data is stored in memory, therefore IO performance is guaranteed. However, memory is powerful, but expensive. It seems not realistic to store large amount of data in memory. Redis gives us its solution of optimization, EXPIRE key. "Set a timeout on key. After the timeout has expired, the key will automatically be deleted. A key with an associated timeout is often said to be volatile in Redis terminology."1 As Redis official document describes, Redis can set a timeout on key, and the key would be deleted after a certain time. Running in memory is one largest difference, compared with other NoSQL.Relatively less data stored and blazing fast speed guarantee the efficiency of data landing. User could have quite a good experience by using Redis as back-end. Social media has a high demand on accessing speed, and the high concurrency becomes one toughest challenge for any social media website.
Other NoSQL, like MongoDB, have their own suitable working areas, which perfect the function of relational database, so does Redis. Redis is great at caching, and working on a small a amount of data. However Redis is much more than a cache.

## 2.3 What is application cashe

Once an application process uses an external data source, its performance can be bottlenecked by the throughput and latency of said data source. When a slower external data source is used, frequently accessed data is often moved temporarily to a faster storage that is closer to the application to boost the application's performance. This faster intermediate data storage is called the application's cache, a term originating from the French verb "cacher" which means "to hide." The following diagram shows the high-level architecture of an application cache:




The cache's main purpose is to reduce the time needed to access data stored outside of the application's main memory space. Additionally, caching is also an extremely powerful tool for scaling up external data sources and mitigating the effects that usage spikes have on them. An application-side cache effectively reduces all resource demands needed to serve data from external sources, thus freeing these resources for other uses. Without the use of a cache, the application interacts with the data source for every request, whereas when a cache is employed only a single request to the external data source is needed, with subsequent access served from the cache.

Additionally, a cache also contributes to the application's availability. External data sources may experience failures that result in degraded or terminated service. During such outages the cache can still serve data to the application and thus retain its availability.

An application's cache is intended for storing data that the application's process requires rapidly during execution. Though each application is unique and can potentially store any kind of data in its cache, an application would typically use a cache for:

Configuration settings: Information that the application requires to make bootstrapping and runtime decisions is often stored in relatively slow storage (such as text files on disk or a shared configuration repository). By keeping cached copies of these settings, the application can access the data with minimal latency.

Localization and internationalization data: User-facing applications often provide localized variants of their interface to accommodate international audiences. Internationalization data is usually stored external to the application so it can be managed separately. Because this data is needed to serve most requests, caching it yields improvements in application response times.

Templates and partially rendered responses: Many applications compose their replies by adding data to templates and pre-prepared content (such as HTML fragments and JSON snippets).

Reusable results of compute-intensive functions: Sometimes an application's workflow requires the generation of resource-intensive results. Once these results are obtained, there are cases in which the results could be later reused, such as when performing partial aggregates. The cache acts as an ideal intermediate medium for retaining such results between requests.

Session data: Caching user session data is an integral part of building responsive applications that can scale. Because every user interaction requires access to the session's data, keeping it in the cache ensures the fastest response times to the application user. Caching session data at the application level is superior to alternative methods. Keeping sessions sticky at the load balancer level, for example, practically forces all requests in a session to be processed by a single app server, while caching allows the requests to be processed by any app server without losing users' states.

DBMS data: Most traditional databases are designed for providing robust functionality rather than speed at scale. The application cache is often used for storing copies of lookup tables and the replies to expensive queries from the DBMS, both to improve the application's performance as well as to reduce the load of the data source.

API responses: Modern applications are built using loosely coupled components that communicate via an API. An application component uses the API to make requests for service from other components, whether inside (e.g. in a microservices architecture) or outside (in a SaaS use case, for instance) the application itself. In cases where an API's reply can be stored in cache, even if only for a relatively short duration, the application's performance is improved by avoiding the inter-process communication.

Application objects: Any object that the application generates and that could be reused at a later time can be stored in the cache. The method for caching objects varies by application, but most caches allow storing both serialized as well as raw object data. A typical example of an often-cached application object is a user profile that is made up of data from multiple sources.

### *2.3.1 Cashe object with Redis*

In this paper, they mainly consider the method which improves the system performance by reading data more frequently from applications and also saves time. Implementation of methods is developed in Nodejs code in the backend provide applications program interface with the redis-client and redis-server.If we work with the system that all the time selling books it is a good idea that those items we get from the database to cashe in Redis in order to have faster access.

```
SET book:12e12e12e12 “{id:'12e12e12e12',title:'Animal
Farm',ISBN:'9780436350306',genderId:
1,authorId:2132}”

GET book:12e12e12e12 // returns the object
```

**Fig 1: Using Redis in-memory cashe bookstore in real example**

We will demonstrate another to get the data from Redis, and to display to the applications.

```
//Add book on redis
await Redis.set(id, book);
await Redis.sadd(`Book:genderId:${book.genderId}`, id);

// get all books by genderId
const bookIds = await Redis.smembers(`Book:genderId:${1}`);
const books = await Promise
.all(bookIds.map(bookId => Redis.get(bookId)));
```

**Fig 2: Displaying the data in redis**

On the first line of the code we want to add the book by id or genderId. this is one way how to implement indexing in redis.
In this case if we every time we will change the name of the book it would be changed also the index.

sein-redis

```
const { Model } = require('sein-redis');
class Book extends Model {
  constructor() {
    super();
    this.modelName = 'Book';
    this.setDefinitions([
      {
        name: 'id',
        primaryKey: true,
      },
      {
        name: 'ISBN',
        uniqueIndex: true,
      },
      {
        name: 'genderId',
        index: true,
      },
      {
        name: 'authorId',
        index: true,
      },
    ]);
  }

  getByISBN(ISBN) {
    return this.getByUniqueIndex({ ISBN });
  }

  getByGenderId(genderId) {
    return this.getByIndex({ genderId });
  }

  getByAuthorId(authorId) {
    return this.getByIndex({ authorId });
  }
}

module.exports = Book;
```

**Fig 3: Working with models and object cashe in redis.**

```
  if (books && books.length) {
    return books;
  }

  books = await this.find({ genderId });

  if (!books || !books.length) {
    return [];
  }

  await Promise.all(books.map(book => bookCache.set(book)));

  return books;
}
```

sein-redis

```
static async getByGenderId(genderId) {
  let books = await bookCache.getByGenderId(genderId);

  if (books && books.length) {
    return books;
  }
```

sein-redis

```
static async getByGenderId(genderId) {
  let books = await bookCache.getByGenderId(genderId);

  if (books && books.length) {
    return books;
  }

  books = await this.find({ genderId });

  if (!books || !books.length) {
    return [];
  }

  await Promise.all(books.map(book => bookCache.set(book)));

  return books;
}
```

### *2.3.2 SQL Cooperated With Redis*

Redis cannot perform well in a real engineering project, but a combo of Redis and SQL, such as MySQL, would incredibly produce a nice teamwork. As previously mentioned, Redis is fast at IO speed, because of its ”caching” feature. MySQL is a popular relational SQL, which has a solid performance and active community. As to a real development project, usually only a small number of data are unstructured, because most of information of world is kind of in a certain structure. Additionally, cloud-computing is growing rapidly, and the local storage would be less essential in the future development. Maybe only the core program and data are running on the local server. If so, Redis will have a much larger role to play in development. Redis has several popular applications in social media nowadays. Moreover, Big Data would be the next breaking point of NoSQL.

## 3. CONCLUSION

In this paper, they make use of an in-memory key-value data storage system is Redis which works on a large data. Redis is blazing fast in speed on reading process as compared to the relational database. The major problem is a previous Client Server framework gives poor execution on read and write process regarding throughput and latency since all servers utilize their own particular memory to deal with the whole procedure, which is time consuming.

NoSQL will be playing a important role in the future development, and has got increasing attention from giant companies and startup. Both academia and industry give a high evaluation of the future of NoSQL. Undoubtedly, NoSQL is not aimed at replacing SQL totally, but assisting relational database work better. Other than this, NoSQL may exploit a definitely new area. Ultimately, we shall always believe the wheel of technology is moving forward.

## REFERENCES


**[1]** Cattell, R., 2011. Scalable SQL and NoSQL data stores. Acm Sigmod Record, 39(4), pp. 12-27.
.

**[2]** DeZyre ”NoSQL vs SQL- 4 Reasons Why NoSQL is better for Big Data applications” https://www.dezyre.com/article/nosql-vs-sql-4-reasonswhy-nosql-is-better-for-big-data-applications/86 19 Mar 2015

**[3]** Zachary Parker,Scott Poe,Susan V. Vrbsky ”Comparing NoSQL MongoDB to an SQL DB ” C3S2E ’13 Proceedings of the International C* Conference on Computer Science and Software Engineering Pages 14-22 Porto, Portugal July 10 - 12, 2013

[4] A. Oussous, F. Z. Benjelloun, A. A. Lahcen, S. Belfkih, “Comparison and Classification of NoSQL Databases for Big Data”

[5] Aaron Schram, and Kenneth M. Anderson. ”MySQL to NoSQL: data modeling challenges in supporting calability.” In Proceedings of the 3rd annual conference on Systems, programming, and applications: software for humanity, pp. 191-202. ACM, 2012.

**[6]** R. Elmasri and S. Navathe, Fundamentals of Database Systems. 7th ed., Chap. 23-24, Pearson Education, 2015.

**[7] Josiah L. Carlson, Salvator Sanfilippo “Redis in Action”**

[8] Stonebreaker, M. 2010. SQL Databases v. NoSQL Databases. Communications of the ACM, Vol. 25, No. 4, pp. 10-11.

**[9]** Itamar Haber – Redis as in-memory applications cashe