ISSN: 2414-1895 DOI: 10.6919/ICJE.202208_8(8).0090

Research and Design of a Small Distributed Key-Value Database based Java

Zhenenji Shao

College of Computer Science, Sichuan University, Chengdu, China

*2427619121@qq.com

Abstract

The main research content of this article is to design and implement a Key-Value database management system that simply runs completely in memory, and to provide it with a simple master-slave mode implementation. It mainly optimizes the threading model of Redis, retains the clever single-threaded data structure of Redis, and reduces the load of consumer threads under the premise of ensuring thread safety. At the same time, the utilization rate of multi-core CPUs is improved and the waste of hardware resources is reduced.

Keywords

Redis Database; Netty Framework; Java Language.

1. Introduction

In recent years, with the continuous generation of massive data and the continuous improvement of hardware configuration, the demand for data query has become more vigorous, and ordinary relational databases have gradually failed in some scenarios. Higher scalability, faster query speed, and richer supported data structures.

The emergence of the Key-Value database management system that runs completely in memory largely meets the needs of users for data query in a high-concurrency environment. Therefore, a completely memory-based Key-Value database that remains robust in a high-concurrency query environment has been warmly welcomed by users. Its representative works mainly include Redis. In production environments, Redis is widely used. In order to take full advantage of multi-core CPUs and support to store more data, multiple Redis instances can be deployed on the same host or a Redis cluster can be formed. This method has a complex structure and high learning cost. For ordinary users, building a cluster is a very challenging task. Once there is a configuration or runtime problem, ordinary users may make mistakes because of their lack of understanding of the cluster. operation with catastrophic consequences. Based on the above background, in order to increase the reliability of the database while making full use of the resources of a single host, this paper provides a simple master-slave architecture implementation for the database. The main contents of the paper are as follows:

(1) The threading model of Redis is optimized, and the single-threading model of Redis is optimized into a producer-consumer model. Under the premise of concurrent access to multiple data structures, the performance will be significantly improved.

(2) The ingenious single-threaded data structure of Redis is retained, and the load of the consumer thread is reduced on the premise of ensuring thread safety.

(3) Improve the utilization of multi-core CPU and reduce the waste of hardware resources.

ISSN: 2414-1895 DOI: 10.6919/ICJE.202208_8(8).0090

2. Key-Value Database Management System

The technologies or tools that this article relies on mainly include ProtoBuf, Netty, Maven, and JDK.

2.1 Database Key

Since this article describes the Key-Value database management system, each key-value pair is composed of objects.

Its corresponding value can be one of five types: ordinary object, list object, set object, ordered set object, and dictionary object.

normal object.

In order to enable the same object to represent strings, integers, long integers, double-precision floating-point numbers, etc., a new class is defined in this paper, and instances of this class are defined as ordinary objects.

dictionary objects.

A dictionary is an abstract data structure used to hold key-value pairs. In a dictionary, you can associate a key with a given value. This database refers to the Map of Redis, and implements the custom class ElasticMap in Java. When expanding or shrinking, this class will move all elements in all slots of the original object to the newly created slot, and optimize it to perform a read operation every time. Then move all elements in one slot of the old object to the new slot, thereby spreading the continuous reHash time among other operations. The object consists of Node, MapObject, ElasticMap.

ordered collection object.

The ordered set object is mainly composed of a skip list (SkipList) and a dictionary. If the skip table is used alone to determine whether the value exists, it may be necessary to traverse the entire skip table. Therefore, the skip table can be combined with a dictionary to optimize the best time complexity for judging whether the value exists. O(1).

List Objects and Collection Objects.

Since JDK provides default implementations, namely ArrayList and HashSet, this article will not elaborate too much.

2.2 Server

2.2.1 Data Serialization and Persistence

In this article, persistence generally refers to the process of dumping the five objects described in Chapter 3 to disk. Since the size of these five objects is very likely to exceed 2G, this is beyond the scope of protoBuf's processing. Therefore, these five objects must be written to disk in a stream.

For object serialization, in this article, it generally refers to the structure required for mutual communication between the client and the server or between the server and the server. Since this article uses ProtoBuf as a serialization tool, this article customizes the protocol format of network communication based on ProtoBuf, as shown in code listing 1.

Code Listing 1 Protocol Format:

```
syntax="proto3";
option java_package = "yourPackage";
option java_outer_classname = "yourClassName";
enum TurtleModel;
enum TurtleValueType;
message TurtleValue;
enum TurtleParaType;
enum ExceptionType;
```

ISSN: 2414-1895 DOI: 10.6919/ICJE.202208_8(8).0090

```
message ResHead;
message ReqHead;
message DataBody;
message TurtleData{
    int64 requestId=1;
    bool beginAble=2;
    bool endAble=3;
    oneof Value{
        ReqHead reqHead=4;
        DataBody dataBody=5;
        ResHead resHead=6;
    }
}
```

2.2.2 Database Threading Model

Based on Netty, this paper designs and implements a custom producer-consumer model.

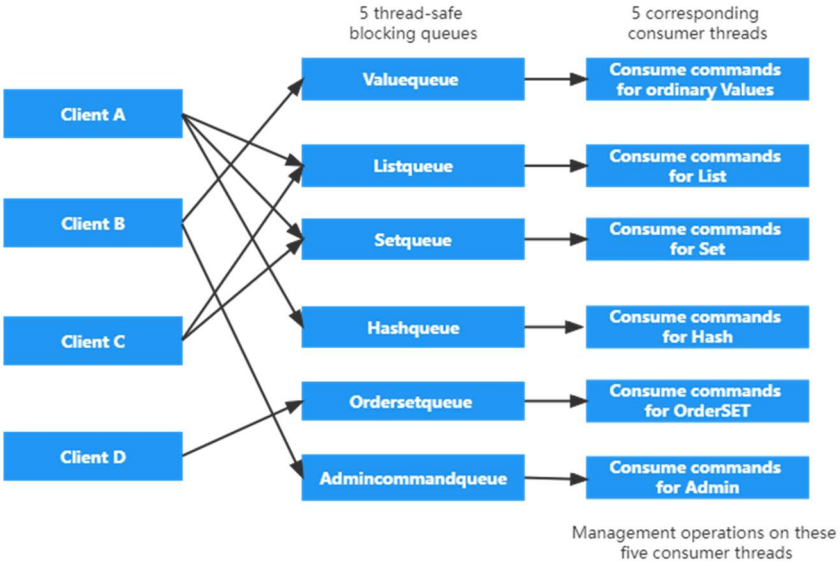

Figure 1. The database threading model described in this paper in a stand-alone environment

As described in Figure1, when the server receives a client command, it deserializes its data, and according to the type of client request, encapsulates it into a Method and the variable parameters required by this Method in the Netty thread . Since this process runs in a single thread, the callback after Netty completes the read operation, this set of operations is thread-safe.

These five blocking queues are an area shared by all threads. Since multiple threads access them, in order to ensure thread safety, this article uses a thread-safe blocking queue in the JUC package. At this time, the consumer thread continuously polls its corresponding blocking queue, takes it out and consumes it, and then writes the consumption result to the client.

The main function of the Admin thread is to process management commands to the database. For example, the client actively requests the server to dump all data, clear all data, check the status of the database or JVM or operating system, perform master-slave synchronization or command propagation and other operations. To sum up, this paper implements this producer-consumer model

ISSN: 2414-1895 DOI: 10.6919/ICJE.202208_8(8).0090

and simplifies the functions of the consumer thread as much as possible to process as much data as possible in limited CPU cycles.

2.2.3 Model Data Persistence

The persistence described in this section generally refers to persisting data to disk. The persisted data can be saved locally or transmitted to other servers for processing by other servers. Since the main content of this article is the Key-Value database, the keys must also be persisted when the five value objects described in Section 2 are persisted. Since the structures of these five value objects are similar, and this article adds expiration time support for ordinary objects, this article only describes the persistence of ordinary objects in detail.

As shown in Table 1, the persistent file of a common object consists of the number of common objects, the number of expiration time keys, and data entities. The data entity contains a entries represented by Table 2.

Table 1. The overall structure of common object persistence

4 bytes represent the number of keys - ordinary objects (a)	4 bytes represent the number of keys with expiration time (b)
data entity	

Table 2. Each constituent item of a data entity

4 bytes represent the key length (n)	n bytes represent the entity content of the key
Entity content of a plain value object	
Expiration ID	8-byte long integer representing timestamp

2.3 Client

Since the user cannot communicate with the server directly, a client must be implemented as an intermediary between the user and the server, and the client must expose a concise and usable API to the user for the user to use.

Now I give a simple case to illustrate how the client interacts with the server.

Code Listing 2 The API that will be called:

Pair<ResHead, Collection<DataBody>> set(String key,TurtleValue value, Long time);

The function of the code in Listing 2 is to insert a key-value pair into the server and set the expiration time. Before the user calls the API, if the client does not establish a connection with the server at this time, the user must provide the IP and port. In this article, since calling the API to start the client is blocking, that is, after the current thread starts the client, the client will be blocked unless it receives a stop command. Therefore, the thread calling this API cannot be the thread that starts the client, and a new thread must be created to establish a connection between the client and the server.

Code Listing 2 encapsulates these parameters into the protocol format described in Section 2.2.1 and sends them to the server. In order to make a one-to-one correspondence between the client's request and the server's reply, a requestId must be carried when requesting from the server, which is a long integer and is thread-safe.

When sent, the current thread blocks waiting for the command to return. At this time, the server converts the protocol format described in Section 2.2.1 into the format of reflection call in Java, and passes it to the consumer thread to run through a thread-safe blocking queue. After the consuming thread runs, it delegates the running result to the Netty thread and returns it to the client.

When the current thread finds that it has received a reply, it will unblock and return to the caller. Pair is a 2-tuple, the first element of which indicates whether the request is successful or the server is

ISSN: 2414-1895 DOI: 10.6919/ICJE.202208_8(8).0090

processing the command, what exception occurred. The second element is a collection that represents the received data entity.

2.4 Master-slave Mode

The master-slave mode generally refers to the structure of a single master server and multiple slave servers in the same database cluster. In most cases, the demand for data query is far greater than the demand for data modification. Therefore, the master-slave mode is usually used to achieve read-write separation. This article provides a simple master-slave mode implementation. Since it does not support automatic master-slave switching, when the master node goes down, the cluster state needs to be restored manually.

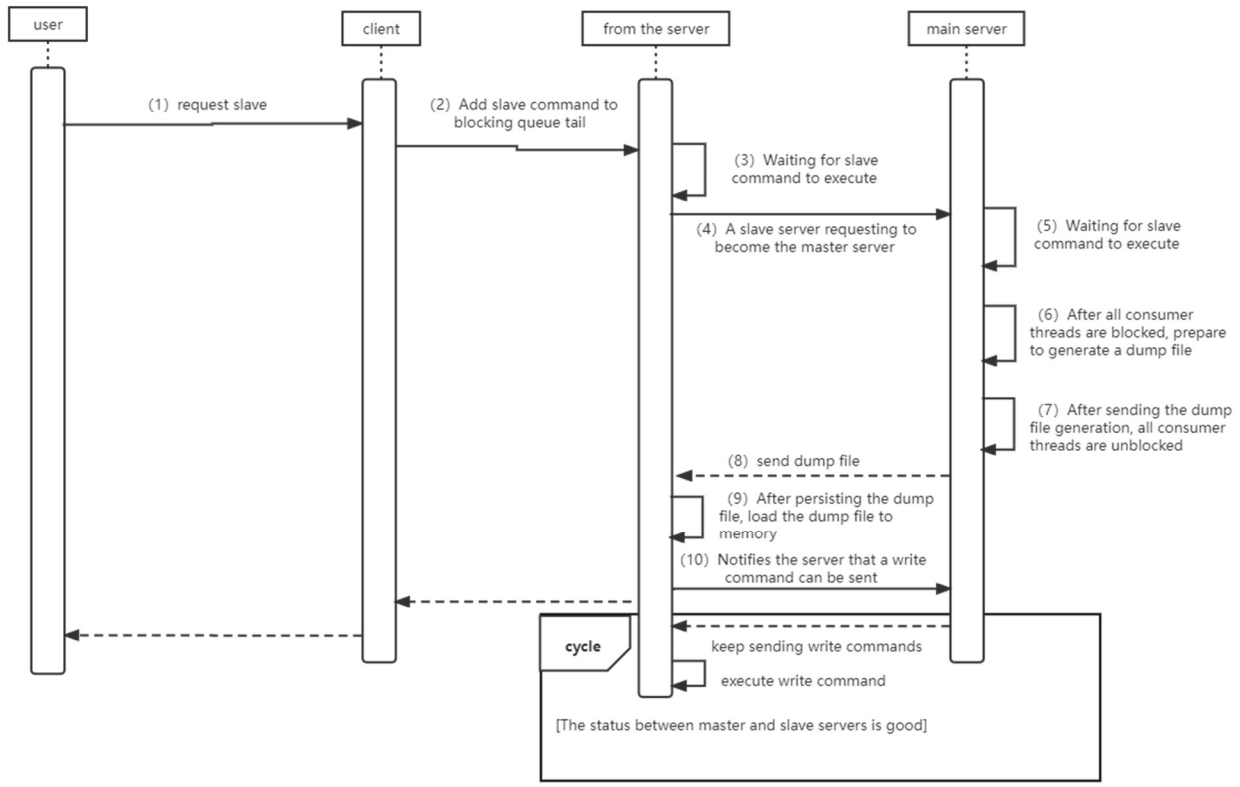

Figure 2. master slave synchronization sequence diagram

(1) After the user activates the master server and the slave server, open the client and call the client's master-slave synchronization API.

(2) The client adds a slave command to the blocking queue of the slave server. Since the slave server has just started, there should be no data in it. If its data is not empty, the slave server is overwritten with the data of the future master server.

(3) Since there is basically no request to the slave 0 server at this time, this command will be executed soon.

(4) The slave server will create a client by itself, request to become a slave server of the master server, and send its IP address and port number to the master server at the same time. At this time, the slave server acts as the client to send data to the server, and acts as the server to process the client's request.

(5) Since the management thread of the main server may execute commands of other clients, this command may be a long time-consuming command. Since only one thread processes management commands, there are no two or more requests for master-slave synchronization on the master server at the same time.

ISSN: 2414-1895 DOI: 10.6919/ICJE.202208_8(8).0090

(6) When master-slave synchronization occurs, all consumer threads of the master server must be blocked (since the management thread is executing the extremely time-consuming instruction of master-slave synchronization at this time, the management thread is also equivalent to being blocked). Different from the server dumping data periodically, or the client actively calling the relevant API to make the server dump data, because the master server data must be in a consistent state, the copy of the current data can be used by the slave server.

(7) Since the operation of dumping these five data structures separately involves the use of both CPU and disk resources, in order to make full use of multi-core processors and speed up the time of data dumping, I use parallel dumping way of data. Since the main thread has to wait for the dump of these data structures to complete before continuing to run down, I used CountDownLatch, a synchronization tool, to synchronize different threads. When all data structures have been dumped, the master server will start to record all instructions that modify the database state and start processing non-administrative commands. Since writes to SSDs are much faster than transfers over the network, in order to get other commands to be processed as quickly as possible, I use the method of first storing the dump data to disk and then transferring it over the network to the slave server. Instead of dumping, streaming to the slave server.

(8) The master server reads the dumped data in the disk and sends it to the slave server. Due to the limited network IO resources, I adopted the method of sending the dump files generated by five data structures to the slave server in turn.

(9) When I received the five dump files of the master server from the server, I took the method of first saving the files to disk and then reading the dump files.

(10) After the slave server reads the dump file into the memory and parses it, it notifies the master server that it can send the cached command to modify the database state to the slave server.

At this point, when the master server receives the request from the slave server, it will enter the command propagation stage.

3. Conclusion

In order to test the performance of the system, I used two Alibaba Cloud cloud servers to test the performance indicators of the system. The client simultaneously initiated ten concurrent requests to the server, each request calling 200,000 for each data structure. times, that is, a total of 10 million requests are sent to the server for each test. After testing, in this case, the QPS of the server can reach about 170,000. When there is data processing, the highest CPU utilization of the server is 65%, because Netty is processing IO events.

In the same client and server, I used the default test tool of Redis for testing, and its performance was about 72,000 QPS. The CPU utilization of single-threaded Redis is about 10%.

After the above design, a simple Key-Value database management system is realized. Under ideal conditions, that is, when the client evenly processes each data structure of the server, the database management system is 2 times faster than Redis, but the CPU utilization rate is 2 times faster. But it is 5 times that of Redis.

References

[1] Zhuang Rong. Design and implementation of a distributed data transmission system based on Netty [J]. China New Communications, 2019, 21(17): 144-145.

[2] Zhen Kaicheng, Huang He, Song Liangtu. IoT data access system based on Netty and Kafka [J]. Computer Engineering and Applications, 2020, 56(5): 135-140. DOI: 10.3778/j.issn.1002- 8331.1811-0261.

[3] Ding Qiang, Tang Lanwen, Ren Nuer, et al. Design and implementation of high concurrent data processing architecture based on Netty [J]. Encyclopedia Forum Electronic Journal, 2018, (20): 781, 787.

[4] [Fan Huafeng. A Design Method of Network Application Server Based on Netty Framework [J]. Fujian Computer, 2015, (10): 33-34. DOI: 10.16707/j.cnki.fjpc.2015.10.016.

ISSN: 2414-1895 DOI: 10.6919/ICJE.202208_8(8).0090

[5] Xia Fei. Research and implementation of message middleware based on Netty [D]. Sichuan: University of Electronic Science and Technology, 2018.

[6] Huang Jianhong. Redis Design and Implementation [M]. Machinery Industry Press, 2014.