Research on mass monitoring data Retrieval Technology based on HBase

Haojie Xu1,a
1Beijing University of Technology, Bejing, China
aemail: Xuhj@emails.bjut.edu.cn

Abstract: With the arrival of the era of big data, the data generated by the monitoring system grows rapidly, the storage and quick retrieval of massive monitoring data pose a challenge to the database. HBase is an open-source version of Bigtable, built on top of HDFS, based design and query on rowkey is quick, but with the development of database application, when retrieving non-rowkey fields, HBase can only query by scanning the full table, which obviously cannot meet the actual requirements, and the query of the monitoring data is mostly based on the most recent data, so for this problem, this article presents a secondary indexing approach based on coprocessor and Redis. This method realizes the rapid creation and automatic update of secondary indexes in HBase and Redis through the coprocessor. It gets the corresponding rowkey quickly during retrieval, and query the corresponding data in the data table according to the rowkey. The former stores the index of the data table, while the latter stores the index of the latest data in the data table, to improve the real-time performance of data retrieval.

1. Introduction
With the continuous improvement of the status of the Internet in daily life, the Internet has become a necessary tool, running through the necessities of people's life. There are monitoring systems in every aspect of daily life. Therefore, an enormous amount of surveillance data will be generated every day, such as video footage, road vehicle picture information. Because of its huge commercial value and market demand, it is becoming a new engine to promote the rapid development of the industry. The storage and retrieval of massive monitoring data also poses severe challenges to the traditional relational database[1]. Because the traditional database can not meet the high concurrent request and storage, the non-relational database arises at the historic moment, among which HBase is the most widely used. HBase is a distributed non-relational database for storing massive data in columns, this technology is derived from the Google paper "Bigtable: A distributed storage system for structured data. HBase uses Hadoop HDFS as its file data storage system, Hadoop MapReduce as its massive data processing system, and Zookeeper as its collaborative service[2]. HBase stores data in the form of a table, which is composed of rows and columns. It divides the columns into several column clusters, while the data of a row can only be identified by a unique rowkey. However, there are still some problems with it, for example, it can only retrieve data through rowkey and Range of rowkey only supports single-row transactions, does not support cross-row transactions, all of which restrict its further development.

In response to this question, this paper proposes a HBase secondary index scheme based on coprocessor and memory database (Redis), for the fields and rowkeys involved in filtering conditions in the HBase data table, the secondary index table is constructed in HBase and Redis through the
coprocessor, and automatic index construction and update is realized through the coprocessor, Redis only stores the index of the latest data in the data table and drops the old data index in time. For example, secondary index information for the latest week's data to reduce disk access overhead and improve real-time access to new data. During the retrieval, the rowkey conforming to the filtering conditions is looked up through the Redis index table or HBase index table, and the Rowkey got in the HBase data table is quickly queried to improve the retrieval efficiency and performance of multi-conditions.

2. Hbase secondary index and Coprocessor research

2.1. Hbase secondary index
The index is a structure to sort the values of one or more columns in a database table, and it is a process to associate the information to be queried with the corresponding record[3]. Although the performance of the HBase database in big data is much better than that of a traditional relational database, some aspects need further optimization. As the primary index of the HBase table, rowkey of each data row uniquely identifies the corresponding data record. Therefore, the query speed based on the primary key is very efficient, however, Hbase does not perform well for multi-conditional queries like traditional databases or queries against specified fields. There are two ways to solve this problem, the first is through a filter provided internally by HBase that retrieves the data in the data table through multiple dimensions during a query. In the case of an enormous amount of data, this method will take a lot of time to scan the entire table, resulting in poor query effectiveness. The second method is to implement the secondary index of the HBase data table, get the rowkey of the corresponding data by querying the index table, and then retrieve the corresponding record in the data table through the rowkey obtained[4].

Currently, there are three fundamental ways to build a secondary index on HBase. The first is a secondary index method based on a third-party independent engine. Examples are ElasticSearch and Solr[5], where ElasticSearch[6] and Solr are full-text search engines based on Lucene. ElasticSearch or Solr[7] can build ElasticSearch or Solr in the index table corresponding to the HBase data table. However, this approach requires additional overhead in maintaining an index cluster built on a third-party engine at all times. The second is memory-based interception methods, such as IHBASE[8] proposed by Yoram Kullbok and Dan Washusen. This method will store the data in the HBase table as another column family, but with increasing data, the data volume of the single table will increase sharply, and frequently cause the split and merge operation of fields. The third is the secondary index method based on the coprocessor, such as Huawei's Hindex, which stores data and index in the data table and index table respectively, and creates and updates indexes through the coprocessor, but this method frequently performs secondary table lookup for new data or data that often needs to be qualified, which causes excessive disk access overhead and does not perform well in real-time.

2.2. Coprocessor
The most prominent disadvantage of HBase as a column family database is that it is difficult to create secondary indexes, and it is difficult to perform summation, calculation, and other operations. Although the HBase database integrates MapReduce in the data storage layer and can be effectively used for distributed computing of data tables, but when doing some simple calculations, it is still desirable to put the calculation process on the server-side, which can reduce communication overhead and achieve better performance. HBase introduced Coprocessors after version 0.92[9], which have two types of observer and endpoint. Among them, the observer is similar to the trigger in MySQL database[10], the main effect is to trigger the dependencies we need for another operation when performing one of the monitored operations, for example, you can listen to the process of adding or deleting the HBase database and perform some specified operations before or after inserting or deleting data into the database. The Endpoint is similar to a stored procedure in a traditional relational database, where the client can call the coprocessor to execute a piece of server-side code and return the execution result
to the client for processing[11]. Introducing the coprocessor enables the client to perform some operations of inserting code, complete the construction and update of a secondary index on the server-side, and thus develop a custom secondary index scheme.

Redis is a fully open-source, BSD-compliant, high-performance key-value database, its data are saved in memory. Meanwhile, Redis can synchronize the data in memory to disk periodically, and load them again for use when rebooting. Moreover, data structures supported include String, List, Set, sorted set, hash, etc. Redis provides extremely high read/write performance, with reads up to 100,000 times per second and writes up to 80,000 times per second, thus providing high efficiency and performance when performing non-rowkey queries.

3. Implement Hbase secondary index scheme based on coprocessor

The HBase secondary index system based on coprocessor and Redis consists of client, index management module, Redis secondary index module, and data processing module. As shown in Figure 1, when the client issues a request to insert, delete or update data according to user requirements, the region of the data table will add this record, and the coprocessor in the index management module will capture the corresponding put/delete/update operation, according to the established strategy. According to formulate strategies to index table insert/delete/update corresponding index entries, for example: insert/delete/update the data for the new data, the coprocessor simultaneously to Redis secondary index table and HBase secondary index table insert/delete/update corresponding index entries, and scan Redis secondary indexes the data in the table, will delete outdated index data. When the deleted/updated data is not the latest data, the coprocessor only deletes/updates the corresponding index entries in the HBase secondary index table.

![Figure 1 System composition diagram](image)

When the client sends a request for query data to the Hbase cluster, as shown in Figure 2, the request is first captured by the coprocessor to determine whether the data being queried is the latest data. If it is the latest data, the corresponding rowkey will be quickly got through Redis multi-condition query in Redis secondary index table. If it is not the latest data, the corresponding rowkey will be queried from HBase secondary index table and returned, then according to the gained rowkey, a quick query is conducted in the HBase data table through rowkey, to improve the query efficiency and timely reduce disk access overhead to improve the performance of the system.
4. HBase secondary index data structure

In the scheme designed in this paper, to fully use the storage characteristics of the HBase database, reduce the number of HBase secondary index tables as far as possible, and improve index performance, all index tables are fused into one index table, and indexes are divided by index identification. Data tables and index tables have the following characteristics:

1. A monitoring data corresponds to a HBase data table.
2. A data table corresponds to a secondary index table.
3. A field in a data table corresponds to an index identifier.

First, in the HBase secondary index table, the starting rowkey of the region of the data table needs to be merged into the rowkey of the index table, when the record of the data table continues to increase and the data in a region exceeds the preset size of the region, a split operation will be carried out and a region sharding will be conducted, the region of the corresponding index table must also make corresponding operations. This is to ensure the distribution synchronization of the data table and index table, so that the process of data retrieval is as localized as possible, with fewer RPC calls. Therefore, the rowkey of the HBase index table is designed as: Region starting rowkey + index identifier + index value.

| Table 1  | data table |
|----------|------------|
| Rowkey   | date       | Content:dir |
| 001      | 2020-01-01 | dir1        |
| 002      | 2020-01-02 | dir2        |
| 003      | 2020-01-03 | dir3        |

Table 1 is the data table in HBase, and the HBase secondary index is established for the date. For example, if the identifier of this data table is IDX1, then the rowkey of the index corresponding to the first data is 001idx120200101. The corresponding HBase index table of the data table in Table 1 in Table 2, the primary key of the original data table is interchanged with the value. The original value acts as the primary key, namely the rowkey of the index, and the original primary key is placed in the value position. The data table model of HBase can be expressed as: R → {Xi(XYi) →Vi} → {Ti},i = 0,1,2,...,n,(1) where R is the primary key, X is the column family, XY is the qualifier of the column, V is the value, and T is the timestamp. The secondary index model of HBase is: Vi → {Xi(XYi) →R} → {Ti},i = 0,1,2,...,n.(2)

| Table 2  | Hbase secondary index table |
|----------|----------------------------|
| Rowkey   | Col                        |
| 001idx120200101 | 001  |
| 001idx120200102 | 002  |
| 001idx120200103 | 003  |
Second, Redis, which stores the latest data index information, can achieve the secondary index in the form of key-value, to achieve higher retrieval efficiency, the form of its key can be expressed as: index identifier + index value, and value stores the corresponding rowkey. For example, the data in the data table with rowkey as 001 can be IDx120200101 and value as 001. The generated Redis secondary index table is:

| Table 3 Redis secondary index table |
|------------------------------------|
| key value                          |
| idx120200101 001                   |
| idx120200102 002                   |
| idx120200103 003                   |

5. Experiment and result analysis
To test the proposed HBase secondary index scheme based on coprocessor and In-memory database (Redis), verify its impact on data writing performance during the construction of HBase secondary index and whether the performance of data retrieval after the construction of HBase secondary index is improved. In this paper, the Hadoop cluster, Zookeeper cluster, HBase cluster, and Redis database is set up on the host.

This article uses insert statements to test, inserting 1W, 10W, 100W data, and testing HBase retrieval and index retrieval of fields that are not primary keys, respectively, to reduce the error of the test and ensure the accuracy of the experiment to the maximum extent, insert and retrieve were tested for 10 times respectively, and the average value of the results was taken. The results of the insert performance test are shown in the following table:

| Table 4 Insert performance |
|----------------------------|
| Insert records 1w 10w 100w |
| No secondary index 6049.3ms 10573.6ms 117254.7ms |
| Secondary index included 7124.4ms 12634.1ms 129583.6ms |

According to the above table, when inserting data into HBase tables with and without secondary indexes, Insertion with a secondary index takes more time, and as the number of records inserted increases, so does the insertion time in both cases. This is because when inserting data into the database, the coprocessor also needs to create and update indexes to HBase and Redis data, respectively, in the case of secondary indexes. Therefore, inserting data with secondary indexes takes more time.

In this paper, when testing the performance of HBase query, 10 pieces of data, 100 pieces of data and 1000 pieces of data were queried from the database by using query statement respectively, and the average value of the test results of 10 times was also taken. It shows the test query performance results in the following table:

| Table 5 Query performance |
|---------------------------|
| Select records 10 100 100 |
| No secondary index 4049.3ms 5573.6ms 7254.7ms |
| Secondary index included 824.4ms 1234.1ms 1883.6ms |

According to the results of the above table, the performance of the query can be significantly improved by using HBase's secondary indexes during the query. This is because standard HBase systems require a full table scan when querying non-primary key fields, which is time-consuming. When the improved HBase queries non-primary key fields, it first gets the rowkey of the data table through the HBase secondary index table, and then conducts the query according to the Rowkey, to
achieve higher query efficiency. And because of the existence of the new data index table in the in-memory database, it will further improve query performance when the new data is queried.

6. Conclusions

To improve the query performance of the HBase database, this paper proposes a HBase secondary index scheme based on Coprocessor and memory database (Redis), when inserting data into the HBase database, the coprocessor is triggered to build secondary indexes in the HBase index table and the Redis index table and update the indexes in Redis, to improve retrieval efficiency, when rowkey is got indirectly on HBase queries non-primary key fields, because the index of the new data is stored in the in-memory database Redis, query performance is further improved for new data with high query frequency. In the future, we will continue to study the cold and hot prediction algorithm of the data, make the cold and hot prediction when the data is inserted, and store its indexes in the corresponding cache index table, to further improve the utilization rate of the memory database and improve the retrieval efficiency of HBase data.

References

[1] Chi Y.P, Yang Y.T, Xu P, Yang J.X. (2008) Design and Implementation of Monitoring Data storage and processing Scheme based on Hadoop [J]. Computer Applications and Software, 35(06):58-63+157.

[2] Xia C.J. (2015) Research on HBase retrieval Speed Improvement based on Coprocessor mechanism [D]. Hunan University.

[3] Zhou W, Liu X, Chen H. (2008) Research on the General Scheme of HBase Distributed Secondary Index [J]. Software Guide, 17(03):187-190.

[4] Cui D, Shi J.X. (2016) Implementation of HBase Secondary Index based on Redis [J]. Software, 5 (11): 64-67.

[5] Paolo M, Michele A, Alessia B, Claudio A, Sandro L.B, Marko M. (2014) High-Performance Annotation Tagging over Solr Full-text Indexes[J]. Information Technology and Libraries, 33(3):.

[6] Li Q, Lu Y, Gong X, (2015) et al. Optimizational method of hbase multi- dimensional data query based on hilbert space- filling curve[C]//Proceedings of International Conference on P2P, 469-474.

[7] Ji Y.M, Zhang N, Yao H.C, Li K, Li H, Liu S.D, Wang R.C. (2019) HOS: Design and Implementation of A Distributed Storage System based on HBase [J]. Journal of nanjing university of posts and telecommunications (natural science edition),39(05):63-71.

[8] Wang W.X, Chen X.S, Wang H.Z, Wu X.S. (2017) A Second-level Index scheme of HBase mass Data based on Solr [J]. Information Network Security, (08):39-44.

[9] Zhu S.J, Lou Y.S, YE F, LI L, Chen Y. Research on HBase Memory Indexing Mechanism based on Coprocessor [J]. Computer Engineering and Applications,202,56(01):98-105.

[10] Guo H, Zhou J.Q, Zhang Y.Y, Guo K. (2019) HBase Two-level index Method based on Coprocessor [J]. Computer engineering and applications,55(21):87-92.

[11] Zou M.H. (2019) Design and Implementation of Lucene based HBase Full-text Retrieval function [D]. Nanjing University.