Advances in Economics, Business and Management Research, volume 30
International Conference on Engineering Management (Iconf-EM 2016)

Research on the Application of Bank Transaction Data Stream Storage based on HBase

Xiaoguo Wang*, Yuxiang Liu and Lin Zhang
School of Electronics and Information Engineering, Tongji University, Shanghai, China
Corresponding author: xiaoguowang@tongji.edu.cn

Keywords: Hbase, secondary indexing, transaction data stream, distribution

Abstract. In the bank transaction monitoring and anti-fraud system, how to manage and analyze massive amount of data efficiently is a great challenge. Compared to RDBMS, NoSQL databases such as HBase, is more attractive in addressing big data problem. In this paper, we proposed an HBase based data storage and retrieval solution method for bank transaction data stream. We constructed table schema based on the user historical behaviours, and we used strategies of row key optimization and table pre-partition to improve efficiency. Also, for those query fields are not in the leading position of the row key, we build secondary index to optimize this kind of query operations. We have experimented and evaluated the performance of this solution. The experiment results demonstrated the improvement in query performance of our solution.

1 Introduction

In a bank transaction monitoring and anti-fraud system, we need to deal with a lot of bank transaction data stream every day. In order to effectively identify suspicious behaviours and take actions to minimize losses, anti-fraud system for this kind of business requires an accurate and comprehensive analysis of data. Therefore, the efficient storage and retrieval of massive transaction data is of great importance.

Traditional data storage solutions based on RDBMS has limitations in dealing with the massive bank transaction data stream. HBase[1, 2], a distributed storage system[3, 4] relies on HDFS[5, 6], is more attractive in addressing this challenge.

In this paper, we proposed a table schema using HBase to store the massive transaction data stream. Also, combined with row key optimization, pre-partition [7] and secondary indexing strategy, we provide an effective way for data storage in bank transaction monitoring and anti-fraud system.

2 Data model

2.1 Design of table schema

In the current monitoring system, data is stored based on transactions, but in reality, a user's transactions often comes from multiple business channels, including mobile banking, online banking, online payment, ATM and POS machines. When it comes to the analysis of the historical behaviours of individual user, such a storage mode often need to do multiple join operations between tables and this will have an impact on performance. To solve this problem, we constructed historical behaviours records based on users, which means put the records belong to one user together in time order. Figure 1 describes the overall architecture of the transaction data stream processing platform.

Copyright © 2016, the Authors. Published by Atlantis Press.
This is an open access article under the CC BY-NC license (http://creativecommons.org/licenses/by-nc/4.0/).

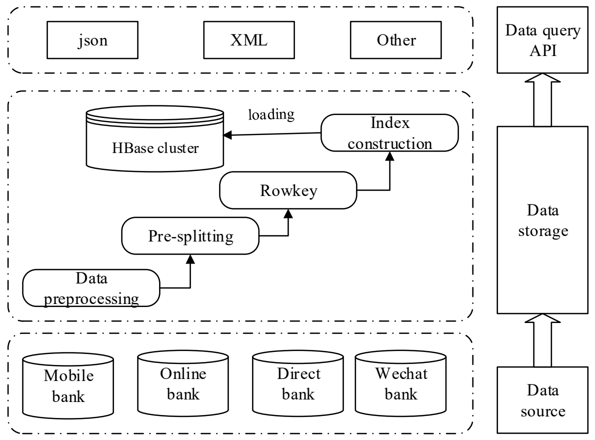

Figure 1. Transaction data stream processing platform

Table 1 shows the table schema of transaction data stream. The basic data storage unit in HBase is a cell, which is identified with the row id, column-family name, column name and the version [8]. The members of the same column-family are stored together physically in HBase, so we can put columns in the same query operation in one column family. Here in the table, we have two column-family: one is named "Data", another is "Index". The "Data" column-family is

Table 1. Table Schema.

Row Key	Column Family: Data		Column Family:Index	
	Qualifier1	Qualifier2 …		
Index-RowKey RegionStartKey-IndexName-IndexKey-Data-RowKey	——		Null	index part
Data-RowKey userID-timestamp	value1	value2	——	data part

mainly used to store transaction records including qualifiers: account ID, receive account ID, transaction amount, currency code, transaction date, transaction channels, etc. "Index" column-family is used for building secondary index. HBase uses row key to retrieve data and it sorts the row keys in lexicographic order. The row keys under this two column-family are composite index with different combination strategies.

2.2 Design of row key

In HBase, row key is critical to the query performance, thus the design of row key is of great importance in table schema. We analyzed the most common query operations in the monitoring system and divided it into three scenarios as it shows in Table 2. Next, we applied different query strategies to these three scenarios. For the most frequently used scene 1, we put the query field into row key to guarantee the best query performance. For moderate frequency scene 2, since it has fixed query mode, we can build secondary index on query fields to improve performance. For other queries, there may be trade-offs between space and time. It is too costly to maintain a specialized index for it since it has the lowest frequency, so we can use HBase Filter to handle this scenario.

Table 2. Data query scenarios.

Scenario	**Query conditions**	**Query frequency**
Scene 1	userID+time range (yyyy-MM-dd HH:mm)	high
Scene 2	bank branch+time range (yyyy-MM-dd HH:mm)	middle
Scene 3	other queries	low

For the row key in data part, we designed it based on the query field in scene 1:

Data-RowKey = userID + timestamp (1)

Since the leading position in row key has the best query performance, we put the most commonly used field user ID in the first place. When doing query operations in scene 1, we can use user ID to identify prefix of row key, thus significantly narrowing the range of the query and quickly get the user historical behaviours records. In addition, we may also need to see records in a certain period of time of that user, so we also put time field in

row key. Setting time range as start key and end key and passing it to an HBase Scanner, we can get the desired searching results within the time range. This kind of row key structure can well supported the basic query requirements without going through a full table scan. What's more, putting time stamp after user ID can make the data more equally distributed among the regions, so as to avoid hot spot caused by the increasingly time series of row key.

2.3 Pre-partition

In default case, when creating HBase table it will automatically create a region. Once inserting data, all HBase clients write to this region until the region is large enough to be split. The splitting of region is automatic and it will have an impact on database performance. To solve this problem, we use pre-partition method. We divided ten regions before head and each region maintains their own start-end keys. Combined with the row key structure mentioned before, writing data can be equally hit these pre-partitioned region and can achieve the load balance within the cluster.

3 Secondary index

The row key structure mentioned before can meet the needs of most queries in monitoring platform. But when it comes to scene 2, the query operations may be time consuming since some query fields are not in the leading position of the key. In this paper, we build secondary index based on coprocessor to optimize this kind of query operation. The solution referred to the existing secondary index program IHBase[9] and hindex[10][11] and it is targeted for bank transaction data stream.

3.1. Design of secondary index

We put the index part and data part in the same region and use different column-family to guarantee physical isolation. Index is designed as an inverted index, converting key-value pairs to value-key pairs. Here, we make the bank branch and transaction time as the combined key and the corresponding data row key as value. The row key in index part as below:

Index-RowKey=RegionStartKey+IndexName+IndexKey+ Data-RowKey (2)

In the first place is the start key of region where data located. This was to guarantee the index and its related data can be stored in the same region. In this way, we can significantly reducing network traffic among the regions and realize localization operations. In the second place is the name of the index, it is used to distinguish different indexes. In the third place is IndexKey, and it is consist of the value of bank branch and transaction time. The last is Data-RowKey, and it is the row key of the designed data. It is used to retrieve data.

In addition, there are two optimization operations for index part: (1) use "-" to connect RegionStartKey and IndexName. This was to ensure that all the indexes will list in front of the data because row key is sorted in lexicographic order in HBase. Also we can guarantee the logical isolation between index part and data part in this way. (2) The storage of HBase is based on column-family, which means that columns in the same family are stored in the same place in hard disk. Assign different column families for index part and data part can ensure physical isolation between them.

3.2 Index construction

In this section, we use coprocessors in HBase to build secondary index. The coprocessor framework provides mechanisms for running your custom code directly on the region servers. An Observer coprocessor is similar to a trigger in a RDBMS in that it executes your code either before or after a specific event (such as a Get or Put) occurs. Observer coprocessors are triggered either before or after a specific event occurs. Using hooks provided by Observer, we can update index when doing Put operation. The construction of index is completed by class MyCorprocessor. It is extends from the class BaseRegionObserver. Override the hook method in this class to ensure that after a Put operation there will be an update in the related index part. The processing procedures are show in Figure 2.

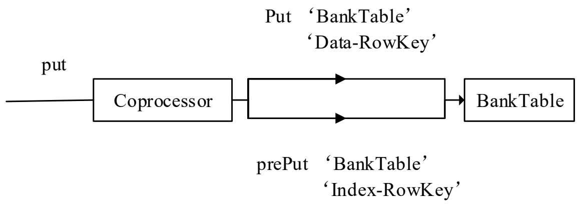

Figure 2. Index construction

Put code and dependencies in a JAR file and place the JAR in HDFS. Using command below to load the coprocessor:

```
hbase alter 'BankTable', METHOD    => 'table_att' , 'Coprocessor' =>
'hdfs:///user/lyx/coprocessor.jar| hbase.bank.MyCoprocessor|1073741823|arg1=1, arg2=2'
```

3.3 Using index

When doing query operations in scene 2, we have three steps:
• Compose the Index-RowKey according to query fields; • Find all qualified Index-RowKey in index part, and cut out the Data-RowKey part from it;
• Go to the data part and fetch data using Data-RowKey.

4 Experimental Analyses

4.1 Experiment Environment

The experiments are performed on a 6 nodes local cluster using one server. Server hardware configuration is: Intel Xeon E5-1620, 64 GB memory, 500GB hard disk. Hadoop version is 2.7.1.2.3.2.0-2950, HBase version is 1.1.2.2.3.2.0-2950, and Zookeeper version is 3.4.6-2950.

4.2 Performance Evaluation

We did performance test on APIs in query scenarios with 8.5 million simulate sample data. For scene 1, we fetch transaction records with given user ID and time range. In order to compare query performance, the test cases are designed to return 1, 10, 100, 1000, 10000 records each. Test Cases and the query retrieval time are shown in Table 3.

Table 3. Scene 1 retrieve time

No.	Test cases	Record number	Retriev e time/s
Scene1_test1	userID+time range (yyyy-MM-dd HH:mm)	1	0.288
Scene1_test2	userID+time range (yyyy-MM-dd)	10	0.290
Scene1_test3	userID+time range (yyyy-MM)	100	0.288
Scene1_test4	userID+time range (yyyy)	1000	0.292
Scene1_test5	userID	10000	0.302

Scene 2 is combination query with bank branch and transaction time. We use secondary index to improve retrieval efficiency. Table 4 shows the retrieval time of 10, 100, 1000 records.

Table 4. Scene 2 retrieve time

No.	Test cases	Record number	Retriev e time/s
Scene2_test1	bankbranch +time range (yyyy-MM-dd HH:mm)	10	0.340
Scene2_test2	bankbranch +time range (yyyy-MM-dd)	100	0.420
Scene2_test3	bankbranch +time range (yyyy-MM)	1000	0.767

Experiments show that the retrieval time from those scenarios are around several hundred milliseconds. The performance can basically meet the requirements of monitoring platform.

5 Conclusions

In this paper, we present a data storage model based on HBase, which can meet the read and write requirements of bank transaction monitoring and anti-fraud system. With strategies of table design, row key optimization and pre-partition, we have improved data retrieval efficiency. Also, we used coprocessor based secondary index to avoid full-scan on large table. With the carefully designed row key, we can put the index and data in the same region to minimize RPC operation delay across regions and use different column-family to ensure physical isolation. Experiments show that this solution has provided an effective way to address the storage and query problem for massive transaction data in bank risk monitoring system.

References

[1] L. George, The definitive guide of Hbase(2013)
[2] F. Chang, J. Dean, S. Ghemawat, TOCS, **26(2)**:205-218(2008)
[3] Y. Zhang, G.Y. Xu, Q. Hua, JOCA, **2015(11)**:3102-3105
[4] Y.H. Ma, X. Meng, L.S. Li, Enterprise Application Development with Hbase(2014)
[5] S. Ghemawat, H. Gobioff, S. Leung, Acm Symposium on Operating Systems Principles (ACM, Bolton Landing, 2003)
[6] K. Shvachko, H. Kuang, S. Radia, Symposium on Mass Storage Systems and Technologies (IEEE Computer Society, Incline Village, 2010)
[7] B. Shen, H.C. Chao, W. Xu, Knowledge Management in Organizations(Springer, Maribor, 2015)
[8] https://hbase.apache.org/book.html

[9] https://github.com/ykulbak/ihbase
[10] https://github.com/Huawei-Hadoop/hindex
[11] http://blog.csdn.net/bluishglc/article/details/31799255