High Performance Secondary Index Design for Complex Queries in Smart Grid System

Qin Jiafeng1, Shu Shitai2, Li Longlong1, Li Chengqi1, Bai Demeng1 and Zheng Wenjie1

1State Grid Shandong Electric Power Research Institute, China;
2Department of Public Health and Management, Bin Zhou Medical University

qinjiafeng0118@163.com

Abstract. Since the traditional relational database is difficult to cope with the massive data and properties produced by thousands of sensors in the smart grid, the HBase database of the new large data platform, as a column oriented key value storage database, has gradually become the mainstream large data platform database. Though HBase can support large data volume, but it is still difficult to cope with the demand of high frequency and short response time in power grid data analysis. In this study, we proposed a query oriented two level index scheme that can speed up the query. The experimental results show that when the two table connection queries are involved, our scheme can provide a minimum of 1.026 to the maximum 4.761 times the acceleration ratio compared to the classic two level index, when it involves three table connection queries, Our scheme provides a minimum of 1.797 to a maximum of 8.581 times the speed ratio of the classic two level index scheme. After further optimization, the scheme can also save much of the storage space of the index table. The two level indexing scheme proposed in this study has a good effect in terms of query performance and storage efficiency.

1. Introduction
The Internet of things [1] and the cross-integration of big data provide new opportunities for the development of power grid technology. The basis of smart power grid is sensor[2] and Internet of things technology[3].All kinds of hundreds of millions of sensors and sensing devices will be around every corner of the smart grid, they will be installed on the important equipment in the smart grid, the important equipment to monitor various parameters. These devices are working day and night to produce data, thus the data table is very large, the number of data table is very numerous. Usually, the various components of a smart grid is associated with each other, which leads to the huge data between tables are almost all related to each other. The relationship between the data table is very complicated. In such a large and large Numbers of data table, complex relationships between these data tables and vast amounts of data, the traditional relational database became helpless. In the era of big data, it must find a new way to storage and manage these data. HBase [4] seems to be a right solution. HBase is a non-relational, column oriented distributed database, distributed database platform, is an open source HBase requirement for hardware is not high, the general commercial computer can build a HBase cluster, it also has the characteristics of storage for a long time, high scalability and the ability to deal with unstructured data. HBase is an appropriate resolution mechanism. In order to check the operation of smart grid, we need to use a lot of data tables for analysis, rather than only one, which can fully reflect the operation state of the grid. In order to
monitor the running state of equipment and timely find equipment fault, these data tables often have to participate in a variety of complex queries. In order to find out the cause of equipment fault, not only that, the query data table time is also required. This kind of data table query needs to be carried out frequently, even though the query is tedious, because the operation safety of power grid should be guaranteed at all times [5]. In the face of such high query frequency and very short query response time, it is very necessary to establish a secondary index for the data table, which can speed up the access to the data table and query. Therefore, how to build a more effective index mechanism for HBase and improve the query efficiency of HBase is a problem that both academia and industry are trying to solve.

In order to improve index access performance and storage performance, we proposed a new index scheme. The main contributions of this paper are:

1) This paper proposes a secondary index scheme for queries, which can build indexes for different queries to speed up the query process.

2) This paper proposes a scheme to save storage space of secondary index, which can save a lot of storage space and make our scheme more practical.

3) This paper has carried out experiments to verify the proposed scheme. This scheme is better than the classical secondary index scheme.

2. Big data storage platform and model

In the smart grid, there are three key factors, such as monitored equipment, monitoring equipment and monitoring time. We can use monitored equipment, monitoring equipment, monitoring time to determine a data record, this is the institute's data model, the basic idea of the details as shown in the literature [6-7].

On the other hand, this model is an event-driven storage model. In a smart grid system, events that can be monitored, and events generate recordable data. The act of generating a unique data record is called an event. An event generates a record in the database, and any event can be uniquely determined by the monitored device, the monitoring device, and the time. Each event in a smart grid system corresponds to a single data record. At the same time, in this event-driven storage model, the minimum number of elements are used to distinguish different data records which can improve the efficiency of data storage and save storage space.

We introduced the smart grid data model. When we want to realize the data model on the HBase model, we can get the event type the information added to the line of key columns and identifier. And different event types are corresponding to different types of data sources, which can use to distinguish data from different sources. There is only one column cluster in each data table. According to literature [8-9], if a data table has more than 10 column clusters, the query speed is very slow when it is queried. The length of any column cluster in the data table should be not exceed 10, and the total number of column clusters in any data table should be not exceed 1000. If a table has too many column clusters, this reduces the performance of the scan operation because it takes too long to find the specified column cluster[10]. In this paper, the column clusters of all data tables are fixed, same and cannot be changed, but the column identifiers can be changed.

3. The design of secondary index for high-performance storage

This paper proposes a two-level index scheme, which can both speed up the query and save the storage space of the index. This scheme is based on HBase storage structure. In general, this second-level index scheme is a query-oriented second-level index technology, and there will be different index schemes according to different queries. The proposed secondary index scheme can effectively speed up the two most basic query operations in the relational database.

For the convenience of narration, two concepts are introduced here, simple query and complex query. Simple query refers to operations such as simple selection operation and simple join operation, while complex query refers to operations such as complex selection operation and complex join operation. Specifically, there is no logical operation in the query condition of a simple query, that is to
say, the intermediate result obtained by querying the secondary index table does not need to perform logical operation, and the intermediate result obtained after querying the secondary index table is the final result. Contrary to simple query, the query condition of a complex query contains logical operation, so the intermediate result obtained by querying the secondary index is not the final result, and the intermediate result obtained needs to perform corresponding logical operation to get the final query result. If you can index simple queries well, you can provide a good foundation for complex queries. In this study, the format of the index table is first defined: the index table is established for select query and join query respectively, and then how to use the index table for data query is discussed. When the storage space is limited, the entire index scheme needs to be optimized to save storage space.

As shown in figure 1, there is only one column cluster in the secondary index table for the selected query, and there is only one column in the column cluster, that is, both the column cluster and the column identifier are invariant. In this column which is established by a unique column cluster and a unique column identifier, it is filled with the rowkey of data table corresponding to the secondary index Table. As we learned earlier in this article, the column values of an index table are 18 bytes long, and in this design, one column in a data table corresponds to one row in an index table. When we want to query a column in a data table, we don't need to scan the entire data table, we just need to quickly access the row key of the index table to get the result we want. When we meet with complex select query, the selecting main involves complex multi-table query and complex selection criteria. The complex queries will be decomposed into several simple query according to the selection criteria, then we can establish a secondary index table for each a simple query. By using the secondary index table for data query, we can get the intermediate results, and these intermediate results are based on complex query logic operation, so as to get the final query results.

Because the secondary Index table for simple select queries is designed for a single table, we call it Separate Index.

In the basic operation of query, in addition to select operation, another operation is join operation. The secondary index table structure designed for simple join query is shown in figure 2.

Because the secondary Index table for simple Join queries is designed for Join operations, we call it a Join Index. If the Join query involves n tables, the corresponding secondary Index table is called the n-join Index.
4. Storage efficient secondary index scheme

To speed up the query, we need to consider indexing all the data table combinations. When $N_{\text{join}}$ is equal to 2, there are two data tables participating in the analysis, if the join query is performed, then only two tables can be included in the join query. Assuming the two tables are $T_1$ and $T_2$, there is only one combination, and we can only join queries between $T_1$ and $T_2$. To facilitate the analysis, we assume that each table has only one column, so to cover the join query between $T_1$ and $T_2$, we only need to have a Join Index table, which is denoted as $I_{(T_1, T_2)}$. Based on the previous discussion of the secondary index table structure for simple join queries, the length of the join index table $I_{(T_1, T_2)}$ is the sum of the length of the data table $T_1$ and the length of the data table $T_2$. When $N_{\text{join}}$ is equal to 3, assuming the names of the three data tables are $T_1$, $T_2$ and $T_3$, there are up to three data tables participating in the join query. This join operation can be performed between $(T_1, T_2)$, $(T_1, T_3)$, $(T_2, T_3)$ and $(T_1, T_2, T_3)$, so this time four data table combinations can be accessed by the join operation. If we assume that each table has only one column as before, then to fully cover join queries between $T_1$, $T_2$ and $T_3$, we need four join index tables, which are $I_{(T_1, T_2)}$, $I_{(T_1, T_3)}$, $I_{(T_2, T_3)}$ and $I_{(T_1, T_2, T_3)}$. And when $N_{\text{join}}$ is 3, the average length of the join index table is longer than when $N_{\text{join}}$ is 2, because the presence of the join index table $I_{(T_1, T_2, T_3)}$ pushes up the average length of the secondary index table.

When $N_{\text{join}}$ is equal to 4, the average size of the secondary index table increases further due to the presence of $I_{(T_1, T_2, T_3, T_4)}$ than when $N_{\text{join}}$ equal to 3. We use NJIT (Number of Join Index Table) to represent the Number of Join Index tables. When $N_{\text{join}}$ is equal to $n$, the join query can be in any 2 data tables, any 3 data tables, any 4 data tables,..., $n$-1 data tables, $n$ data tables. The join index table needs to be overridden by any two data tables, any three data tables, any four data tables,..., any $n$-1 data tables, even any $n$ data tables, then NJIT calculation is as follows:

$$NJIT = C_2^n + C_3^n + C_4^n + \cdots + C_{n-1}^n + C_n^n = 2^n - n - 1 \tag{1}$$

Where is the number of all combinations of five elements taken from three different elements, that is, even in the simplest case where there is only one column per table, we need a join index table to cover all possible join queries. Participate in the more simple connection query data table, the query the corresponding secondary also the longer the length of the index table, even if we do not consider the growth of the secondary index table length, the number of connection index table will increase exponentially with the increase of, that is to say, the index table occupy storage space also will exponentially with the increase of at least, this is unacceptable in practice. Therefore, we must improve the original join index scheme, focusing on its storage optimization.

In order to solve the problem that the storage space occupied by the secondary Index table designed for Join query increases exponentially with the increase of $N_{\text{join}}$, we decided to use Separate Index and 2-join Index to optimize the storage space of the previously mentioned secondary Index scheme. Double connection index table can accelerate the process of query, and won’t take up too much storage space. The length of the double-joined index table is relatively short, just the sum of the two data tables. When $N_{\text{join}}$ increases, a double connection index table storage occupied space does not have significant changes. The storage space occupied by a double-joined index table does not change significantly, that is, as the number of double-joined index tables increases, the storage space occupied by them increases linearly. Therefore, we decided to use a separate index table and a double join index table to cover all possible join queries.
Let's assume that there are $N_{\text{join}}$ data tables involved in the discussion. For simplicity, we assume that each data table has only one column. Firstly, we set up a Separate Index for each table of data. Because each table of data is likely to be accessed by a select query, we need to speed up this common query process. Then we create a two-join Index for all combinations that contain two data tables, that is, a simple Join query secondary Index table for two data tables. If there are $N_{\text{join}}$ data tables, you need to set up $C^1_{N_{\text{join}}}$ a separate index table and $C^2_{N_{\text{join}}}$ double connection index table, where $C^m_n$ is the number of combinations of $m$ elements from $n$ different elements. Next we need to establish $N_{\text{join}}$ a separate index table and $N_{\text{join}} (N_{\text{join}} - 1)/2$ double connection index table. Altogether, there are $N_{\text{join}} (N_{\text{join}} + 1)/2$ secondary index tables. And the length of each secondary index table does not exceed the length of the two data tables combined. At last, the creation of the index table is complete.

5. Experiment and result analysis

In the experiment, we mainly control two parameters to carry out the experiment. These two parameters are Record Number and Ratio. In the experiment, the number of records ranged from 100 to 50,000, accounting for a range of 0.1 to 0.9. In this experiment, the Record Number (record #) is 500, 2000, 10000 and 50000, and the Ratio is 0.1, 0.25, 0.5, 0.75 and 0.9. We expand the join query for the column of Year and compare the query acceleration effect of different index schemes.

We compared the secondary index scheme in this paper with the classical secondary index scheme (such as HIndex, etc.). We can get the acceleration ratio of the proposed secondary index scheme when facing the join query of two tables compared with the classical secondary index scheme. We first draw the change of the acceleration ratio with the number of records. The specific results are shown in figure 3.

![figure 3. The speedup of the secondary index scheme compared with the classical secondary index scheme in the face of a two-table join query](image)

We can see in figure 3, compared with the classical secondary index, the secondary index scheme can provide better speedup, under the condition of minimum it can provide the acceleration of 1.026 times, and under the biggest cases it can provide 4.761 times speed. Even compared with the classical secondary index scheme, the acceleration effect is obvious, and the scheme proposed in this paper is superior.
We made a simple join query on the three data tables to obtain the acceleration ratio of the proposed secondary index scheme compared with the classical secondary index scheme when facing the join query of the three tables. The change of the acceleration ratio with the number of records is shown in figure 4.

![Speedup over Record Number](image)

**figure 4.** The speedup of the secondary index scheme compared with the classical secondary index scheme in the face of a three-table join query

We can see from figure 4, compared with classical scheme of secondary indexes, this paper proposed a secondary index scheme can provide very good speedup, under the condition of minimum can it provide 1.797 times speedup, one of the biggest cases can provide 8.581 times speedup, speedup increases along with the increase of number of records. Even when compared with classic secondary index schemes, the acceleration effect is obvious. In addition, by comparing, it can be found that under the same number and proportion of records, the acceleration ratio of three data tables is greater than that of two data tables. In other words, with the increase of data tables participating in query, the performance of the secondary index scheme proposed in this paper will be better and better.

6. Conclusion

This paper proposes an effective index mechanism and a new type of big data platform to meet the demand of high query frequency and short query response time of smart grid big data. Then for this data table structure, we design a secondary index scheme for select query. A second index scheme for join query is designed. To solve the problem that too many index tables are needed to cover all queries, we optimized the original join index scheme, the index table storage space from $O(2^n)$ reduced to $O(n^2)$. In this paper, we get a storage efficient secondary index scheme, and the experiment proves that our scheme is much better than the classical secondary index scheme. In the experiment, compared with the classical secondary index schemes such as HIndex, the scheme proposed in this paper can provide the acceleration of 1.026 times in the minimum case and 4.761 times in the maximum case. In the case of three tables, compared with the classical secondary index scheme, the scheme proposed in this study can provide the minimum acceleration of 1.797 times and the maximum acceleration of 8.581 times. The more data tables there are, the more obvious the acceleration effect will be. We also discussed the storage performance of this solution, which can save more than half of the storage space with just four data tables. Experiments show that the index scheme proposed in this study can provide good performance.
References

[1] Atzori L, Iera A, Morabito G. The Internet of Things: A survey[J]. Computer Networks, 2010, 54(15):2787-2805.

[2] Gungor V C, Lu B, Hancke G P. Opportunities and Challenges of Wireless Sensor Networks in Smart Grid[J]. IEEE Transactions on Industrial Electronics, 2010, 57(10):3557-3564.

[3] Li L, Hu X, Ke C, et al. The applications of WiFi-based Wireless Sensor Network in Internet of Things and Smart Grid[C]. Industrial Electronics and Applications. IEEE, 2011:789-793.

[4] Apache Group, Apache HBase [OL]. http://hbase.apache.org/, 2017.

[5] Wang C, Zhu Y, Ma Y, et al. A Query-oriented Adaptive Indexing Technique for Smart Grid Big Data Analytics[J]. Journal of Signal Processing Systems, 2017(4):1-13.

[6] Bajda-Pawlikowski K, Abadi D J, Silberschatz A, et al. Efficient processing of data warehousing queries in a split execution environment[C]. ACM SIGMOD International Conference on Management of Data, SIGMOD 2011, Athens, Greece, June. DBLP, 2011:1165-1176.

[7] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters[C]. Conference on Symposium on Operating Systems Design & Implementation. USENIX Association, 2008:10-10.

[8] Kornacker M, Behm A, Bittorf V, et al. Impala: A Modern, Open-Source SQL Engine for Hadoop. [C]. conference on innovative data systems research, 2015.

[9] Huawei Inc. Huawei HIndex [OL] (2014). https://github.com/Huawei-Hadoop/hindex.

[10] Liu B, Zhu Y, Wang C, et al. A Versatile Event-Driven Data Model in HBase Database for Multi-source Data of Power Grid[C]. IEEE International Conference on Smart Cloud. IEEE, 2016:208-213.