A Novel Big Data Index Architecture for Programming Environment

Peng Cao, Haitao Xu
School of Computer and Communication Engineering, University of Science and Technology Beijing, Beijing, China

Abstract—With the rapid development of computer information technology and the upgrading of programming software, the types and number of project codes are growing rapidly, showing typical characteristics of large data such as massive, instantaneous, diverse and variable. The distributed column storage database HBase based on the Hadoop big data platform, has the characteristics of high reliability, high performance, column-oriented, and scalability. It also has good scalability, can store more than ten billions of data, and is suitable for large-scale data reading and writing, which takes advantages in processing large-scale unstructured and semi-structured software data on the programming site. So, this paper studies the big data index architecture in the programming field. In view of the low efficiency of HBase non-primary key attribute query, the inverted index is a natural platform for cloud-based big data storage and query in this large development site. Based on this advantage, this paper designs and optimizes the secondary index architecture based on the HBase for programming field big data inverted index.

1. INTRODUCTION
With the popularization of the Internet, artificial intelligence, 5G technology and various digital terminal devices, a world of interconnected things is taking shape. At the same time, with the explosive exponential growth of data, digitization has become the foundational force for building a modern society, and is driving us to an era of profound changes. People are generating big data at an unprecedented rate. According to the report released by the Internet Data Center (IDC) "Data Age 2025" [1], the global data storage capacity will reach 50ZB by 2020, which is 14% higher than the previous estimate. The data generated each year will increase from 33ZB in 2018 to 175ZB, which is equivalent to generating 491EB of data per day. According to a report released by the international authority Statista in August 2019, the global big data market revenue is expected to reach $56 billion by 2020, an increase of about 33.33% from the expected level in 2018, and the market revenue scale doubled compared with 2016.

At present, software development is becoming more and more intense, and the software development site is becoming larger and larger. It is generally required to support more than 1,000 people. The standard source code sample library covers more than 100 software development projects, and the code size of each project is not less than 100,000 lines, a total of more than 100 million lines of code, a huge amount of code. As the main body of software big data, unstructured data is also developing at a high speed, which poses new challenges to traditional unstructured data management.

The code file data in [2] is mostly small files in different projects. If it is stored directly in HDFS, because the NameNode in HDFS stores the index of all DataNode data blocks, the client accesses the data It is the existence of accessing the actual data in the NameNode first, if it exists. Then the
NameNode node returns the actual storage location of the data in the DataNode to the client. The index file size of each DataNode in the NameNode is 64 bytes. If a large amount of small software data is stored in the DataNode, the metadata information in the NameNode will be so much, which will bring great load to NameNode.

Based on the characteristics of software data in the programming field of the cloud platform, this article chooses to store the software data in the distributed storage system HBase. Based on Google BigTable, HBase [3] is the most widely used key-value NoSQL database, built on the Hadoop distributed platform, and is a highly reliable, high-performance, column-oriented, and scalable distributed NoSQL platform. The design goal of HBase is to solve the challenges faced by relational databases when dealing with unstructured data. It can organize data through row keys, support single-row transactions, and manage unstructured or semi-structured hash data. It is built on HDFS and supplements HDFS functions. It can effectively store and manage massive data in a distributed manner, providing real-time read and write and random access for semi-structured and unstructured massive data. Hbase was designed from the beginning to meet the massive data storage of TB to PB level and high-speed, massive concurrent reading and writing needs [4], these data can be distributed on thousands of cheap commercial servers.

2. DATABASE INDEXING ALGORITHM

Index [5] is the most common and effective method to improve data query efficiency. The essence of indexing is to increase the speed of information query and retrieval. Before the invention of the computer, the idea of indexing has penetrated into our lives. The full-volume encyclopedia can be regarded as an indexing strategy, and indexing can make the query process more efficient and convenient. Therefore, after the advent of computer technology and database technology, the index technology on the database also came into being and flourished.

Traditional database index technology can be divided into clustered indexes and non-clustered indexes according to the physical structure of data storage. Clustered indexes are in order according to the physical location of data storage, and non-clustered indexes are different; clustered indexes can improve The speed of multi-row retrieval, non-clustered index is very fast for single-row retrieval. The index data structures commonly used in traditional database indexing technologies mainly include HASH index [6] and B + tree index [7].

The fundamental change of big data storage methods requires that the indexing mechanism of big data must meet the requirements of supporting multiple queries, efficient retrieval and easy maintenance. In order to solve the big data query processing problem, a new index structure needs to be established for the big data environment. According to the different index structure, the index technology [8-10] is divided into inverted index, secondary index, double-layer index, multi-dimensional index, Hadoop-based index technology.

3. DATA SEGMENTATION PROCESSING BASED ON LUCENE SOFTWARE

Lucene is an open source full-text search engine toolkit of the Apache Software Foundation. It is a full-text search engine [11] architecture that provides a complete query engine and index engine, as well as some text analysis engines. The purpose of Lucene is to provide software developers with a simple and easy-to-use toolkit to facilitate the realization of full-text search [12] in the target system, or to build a complete full-text search engine based on this.

The programming field software data is English source code. Compared with the Chinese word segmentation, the basic unit of the English language is the word, so the programming field software data has a natural advantage over the Chinese word segmentation. The software data segmentation only requires three steps: 1) Separate spaces, symbols, paragraphs to get word groups; 2) Filter to exclude stop words; 3) Extract stems.

This article uses the Lucene custom tokenizer. Lucene has a built-in core API for tokenizer component-the analyzer (org.apache.lucene.analysis.Analyzer), which is used to build a tokenizer
processor that truly performs tokenization on text and there is only one extensible abstract method in the Analyzer class:

```java
protected abstract TokenStreamComponents createComponents(String fieldname)
```

This method should be overloaded when expanding its own analyzer. The parameter fieldName of the method is changed to indicate the field name. Different fields have different processing methods, according to the field name to distinguish. TokenStreamComponents is an internal class that provides two constructors:

```java
public TokenStreamComponents(final Tokenizer source, final TokenStream) {
    this.source = source;
    this.sink = result;
}

public TokenStreamComponents(final Tokenizer source) {
    this.source = source;
    this.sink = result;
}
```

From the parameters of the above constructor, at least one Tokenizer parameter must be provided. The TokenStream class is responsible for word segmentation and processing of the input text, and each item separated by the word segmentation is called a token. In fact, TokenStream has two types of subclasses for word segmentation and processing. One is Tokenizer tokenizer, which completes the separation of input items from the input reader character stream, and the other is TokenFilter, an item filter, which performs feature processing on the divided items. TokenFilter is the decorator mode used. If you need to perform various processing on the word segmentation, we only need to wrap it in layers according to the business processing order.

There is an abstract method in TokenStream as follows:

```java
public abstract boolean incrementToken() throws IOException;
```

This method should be implemented when implementing the tokenizer to tell the tokenizer rules and processing rules. When the return value is false, the word segmentation ends.

TokenStream inherits AttributeSource, this method is used to store Attribute, and provides the corresponding setting and value method. After using Tokenizer and TokenStream to divide the words, and after processing each item, each item will generate corresponding information, such as the text of the item and the position. These are to be stored, then this information is stored in Attribute, AttributeFactory is used to create attribute factory method, no need for us to create.

The process of software data word segmentation is shown in Figure 1:

### 4. HBASE ARCHITECTURE AND DATA STORAGE FORMAT

#### 4.1 HBase architecture

HBase is short for Hadoop database, which is based on Hadoop database and is a NoSQL database. It is mainly suitable for random real-time query of massive detailed data (billions and tens of billions), such as log details, update codes, and historical search. Under the tide of massive data, the use of a non-relational database designed for massive heterogeneous data can solve the needs of large-scale programming sites such as storage and calculation of big data to a certain extent. The commonly used non-relational database HBase [13] has good horizontal scalability and good write operation performance, making it suitable for the storage of heterogeneous, diverse, and massive software data in the programming field.

The HBase cluster architecture consists of the main components in several big data frameworks such as Client, Zookeeper, Master, HRegionServer, and HDFS.
4.2 HBase data structure

The row data in HBase is the only unit determined by \{Row Key, Time Stamp, column Family: column\}. The data in the row is not typed, all is stored in byte code.

(1) Row Key

HBase and nosql databases are the same, Row Key is the primary key used to retrieve records, is the unique identifier of the row in HBase. There are only three ways to access the rows in HBASE table: 1) access through a single Row Key; 2) range through Row Key; 3) full table scan.

(2) Time Stamp

In HBase, it is determined by Row Key and Columns that a storage unit is called a cell. Each cell holds multiple versions of the same data. The type of time stamp is a 64-bit integer. If the application wants to avoid data version conflicts, it must generate unique timestamps by itself. In each row of data, different versions of data are sorted in reverse chronological order, that is, the latest data is ranked first, and the TIME Stamp timestamp records are prepared for future cold and hot data storage.

(3) Column Family

Each column in the HBase table belongs to a column family. Column families are part of the table's schema, and columns are not, and must be defined before using the table. Column names are prefixed by column family. For example, python: kreas and python: numpy belong to the column family of python.

The HBase data model consists of a logical model and a physical model, as shown in the following table:

(1) Logical model

| Row Key | Time Stamp | Project |
|---------|------------|---------|
|         | tn         | Class1  |
|         | tn-1       | Class2  |
|         | ......      | ......   |
| W       | t1         | ......   |
|         | p1…px      | Classk  |
|         | p1…py      |         |
|         | p1…pz      |         |

The above table gives the ideal logical model of HBase. The row record in the table is W, representing a certain keyword, and there are k column families in the table, representing the frequency of W keywords in a certain category under a project.
Step 1: public interface MyCharAttribute extends Attribute: Implement custom word segmentation attributes

Step 2: public class MyCharAttributeImpl extends AttributeImpl implements MyCharAttribute: implement interface of the step 1

Step 3: public class MyTokenizer extends Tokenizer: implement your own tokenizer

Step 4: public class MyTokenFilter extends TokenFilter: definition word processor

Step 5.1: Obtain the item attribute object you want from tokenStream to store item information

Step 5.2: Call the reset method of tokenStream, Reset

Step 5.3: Cyclically call tokenStream's incrementToken method until it returns false

Step 5.4: Take out the attributes you want for each sub-item in the loop

Step 5.5: Call the end method of tokenStream and accept the processing

Step 5.6: Call the close method of tokenStream, Free up resources

Figure 1 Software data segmentation process
(2) Physical model

| Row Key | Time Stamp | Column family | Column | Value |
|---------|------------|---------------|--------|-------|
| W       | tn         | project       | Class1 | p1…px|
| W       | tn-1       | project       | Class2 | p1…py|
| ......   | ......      | ......         | ......  | ......|
| W       | tl         | project       | Classk | p1…pz|

From the logical structure table, the logical storage is still composed of many rows, but the actual storage in HBase or the physical model table is stored in columns, as shown in the table above.

5. DESIGN OF SECONDARY INDEX BASED ON HBASE INVERTED INDEX

5.1 Design of inverted index based on HBase

In this paper, an inverted index is established for software data based on the word frequency in the software data file. The inverted index is similar to the expansion of WordCount in hadoop. It needs to count the number of times a word appears in multiple different software data files. The MapReduce computing framework establishes an inverted index for software data. Since Lucene's Analyzer class has been used to perform word segmentation on the software data, in the MapReduce phase, only the file after word segmentation needs to be indexed.

This article establishes an index to convert the basic format of (term, (funName, tf) to ((term, funName), tf), so that the number of keys with the same key value can be reduced. The pseudo code flow in the Map phase is as follows:

```java
// Rewrite the map () function, the value of the map function parameter is the file after the word segmentation by Analyzer
    String line = value.toString();//retrieve data
    String[] fields = line.split("\|"));//Take | cut each line
    // Get file name only
    FileSplit inputSplit = (FileSplit)context.getInputSplit();
    String pathStr = inputSplit.getPath().toString()
    Int index = pathStr.lastIndexOf("/");
    String strFileName = pathStr.substring(index+1);
    //Write out key= field + "->" + strFileName,value=1
    for (String field: fields){
        context.write(new Text(field + "->" + strFileName), new Text("1"));
    }
```

In order to reduce the output of Mapper, thereby reducing the transmission overhead and storage overhead of Mapper to Reducer, it is a good way to use Combiner in the map phase. The parent class of Combiner is Reducer, which is equivalent to performing a Reducer after each Mapper to set the same key value. The value of is accumulated to get the word frequency of a word in the document. Because the Map stage is distributed.

If the output of the combined word frequency is directly used as the input of the Reduce process, you will face a problem during the Shuffle process: all records with the same word (composed of words, funName, and word frequency) should be processed by the same Reducer, but the current The key value cannot guarantee this, so the key value and value must be modified. This time, use words as key values, funName and word frequency to form value values (such as "math.py: 1"). The advantage of this is that the default HashPartitioner class of the MapReduce framework can be used to complete the Shuffle process and send all records of the same word to the same Reducer for processing. The Combiner pseudocode is as follows:

```java
// Statistics word frequency
    Int sum = 0;
    for (Text value: values){
```
sum += Integer.parseInt(value.toString());
int splitIndex = key.toString().indexOf("->");
//Reset value
Value.set(key.toString().substring(splitIndex) + "->" + sum);
//Reset key
key.set(key.toString().substring(0, splitIndex));
context.write(key, value);// Write out

In the Reduce stage, we need to merge the same key value to generate a document list. For the specific process, see the following code:

```java
Text result = new Text();
// Generate document list
String fileList = new String();
for (Text value : values) {
    fileList += value.toString() + ";");
}
result.set(fileList);
context.write(key, result);
```

5.2 Design of a secondary index architecture based on HBase non-primary key

HBase is a column-oriented memory. The current HBase only supports primary index based on RowKey rows, which itself does not have the secondary index function based on column query. This article aims to design a secondary index architecture for HBase based on the inverted index. The secondary index architecture diagram is shown in Figure 2:

![Figure 2 Secondary index architecture](image)

The secondary index data query process is shown in Figure 3:

![Figure 3 Secondary index data query process](image)
Quickly filter out the primary row key of the data that meets the conditions through the query of the non-primary key, and then quickly query the required data according to the HBase primary row key, and realize the non-primary key secondary index of the inverted index.

6. SUMMARY AND OUTLOOK
This article gives a brief introduction to the massive heterogeneous big data emerging in the programming field environment in the current big data era and cloud platform. Using the Analyzer in Lucene to explore and segment the software data, it is proposed to use cloud-oriented distribution Columnar HBase to store data, use MapReduce to design the inverted index of the text after word segmentation, and finally quickly filter out the main row key of the data that meets the conditions through the query of the non-primary key, and then quickly query according to the HBase primary row key. The data realizes the non-primary key secondary index of the inverted index.

The index structure designed in this paper is only applicable to the indexing and querying architecture of programming field big data. When this system is used to store other non-programming field big data types, it cannot be directly applied. It has certain limitations. Research more general indexing and query algorithms for different scenarios of big data to solve the challenges of the big data era.

ACKNOWLEDGMENT
This work was supported by the National Key R&D Program of China (No. 2018YFB1003905).

REFERENCES
[1] SEAGATE. The Digitization of the World From Edge to Core[J]. An IDC white Paper. 2018.
[2] C. Xu, X. Du, Z. Yan, and X. Fan. ScienceEarth: A Big Data Platform for Remote Sensing Data Processing. Remote Sens. 2020, 12(4): 607.
[3] F. Tianyi, L. Yan, and Z. Ma. Storing and querying fuzzy RDF(S) in HBase databases. International Journal of Intelligent Systems. 2020, 35(4): 751-780.
[4] S. Magdy, Y. Abouelseoud, and M. Mikhail. Privacy preserving search index for image databases based on SURF and order preserving encryption. 2020, 14(5): 874-881.
[5] M. Hayati, A. Setyanto. Index Effect on Data Manipulation Toward Database Performance. 2018, 1140(1).
[6] R. Grycuk, P. Najgebauer, and M. Kordos, et al. Fast Image Index for Database Management Engines. 2020, 10(2):113-123.
[7] Sun, Feng Dong, Q. Guo, and L. Wang. Study and Optimization of T-Tree Index in Main Memory Database. Applied Mechanics and Materials. 2013, 427-429: 2531-2535.
[8] Jin, Yanzhong, and X. Bao. An Efficient XML Index for Keyword Query with Semantic Path in Database. Journal of software. 2010, 5(10): 1052-1059.
[9] K. Narinder, S. Taruna. An efficient approach to optimise I/O cost in data-intensive applications using inverted indexes on HDFS splits. 2019, 15(1-2).
[10] Y. Jin, X. Bao. An Efficient XML Index for Keyword Query with Semantic Path in Database. 2010, 5(10):1052-1059.
[11] Hirsch, Laurence, and T. Brunsdon. A Comparison of Lucene Search Queries Evolved as Text Classifiers. Applied Artificial Intelligence. 2018, 32(7-10): 768-784
[12] L. Fu, X. Zhang, C. Hua. Research on Full-Text Retrieval Based on Lucene in Enterprise Content Management System. 2014, 3468:1950-1953.
[13] Dillon, Chrimes, and Z. Hamid. Using Distributed Data over HBase in Big Data Analytics Platform for Clinical Services. Computational & Mathematical Methods in Medicine. 2017(2017):1-16.