Intelligent Automation & Soft Computing
DOI:10.32604/iasc.2022.022571
Article

Tech Science Press

Fast Access and Retrieval of Big Data Based on Unique Identification

Wenshun Sheng[1,*], Aiping Xu[2] and Shengli Wu[3]

[1]Pujiang Institute, Nanjing Tech University, Nanjing, 211200, China
[2]School of Computer, Wuhan University, Wuhan, 430072, China
[3]School of Computing, Ulster University, Belfast, BT370QB, United Kingdom
*Corresponding Author: Wenshun Sheng. Email: gisma@qq.com
Received: 11 August 2021; Accepted: 18 October 2021

Abstract: In big data applications, the data are usually stored in data files, whose data file structures, field structures, data types and lengths are not uniform. Therefore, if these data are stored in the traditional relational database, it is difficult to meet the requirements of fast storage and access. To solve this problem, we propose the mapping model between the source data file and the target HBase file. Our method solves the heterogeneity of the file object and the universality of the storage conversion. Firstly, based on the mapping model, we design "RowKey", generation rules and algorithm. Then according to the mapping rules of data file fields with the HBase table column, the data in the data file are transformed into HBase. Finally, the retrieved keywords in "RowKey" are stored and used to achieve fast data retrieval by prefix matching or keyword matching method. Our method has been applied to different projects, which shows these results can be applied to the data conversion from regular row store data file to HBase distributed large data storage and has strong commonality. The method can be widely used in HBase big data storage applications.

Keywords: Big data; row store; RowKey; fast retrieval; HBase

1 Introduction

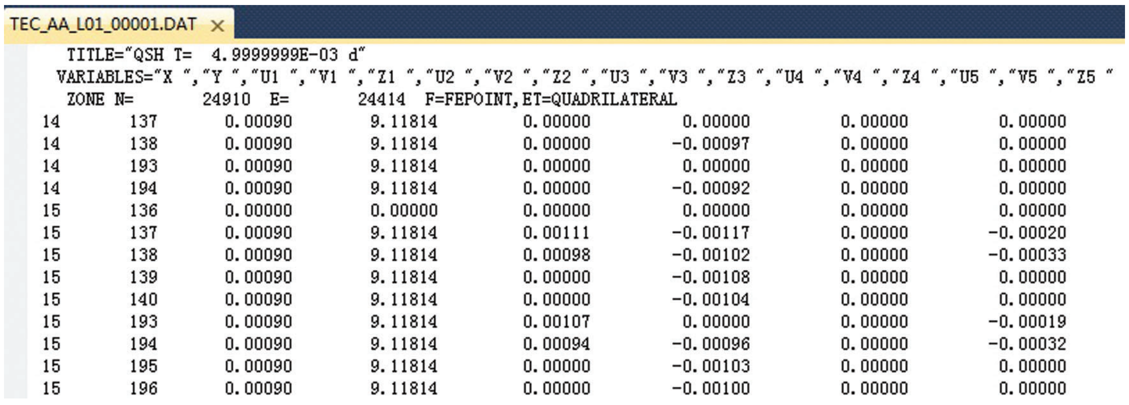
For big data applications, massive data are stored in files by rows [1,2]. With the continuous development and application of distributed database technology, converting these data files into distributed storage can provide a more convenient application environment [3,4]. However, there is only a little literature on the data file conversion to the distributed database. Currently, there are three ways to integrate different types of data migration into distributed databases (such as HBase) [5], using the HBase interface to write special programs to achieve data docking, using tools such as bulk load to complete data import, and writing MapReduce programs to import data into HBase [6–9]. The three methods all have problems such as poor versatility, limited environment, and high operational complexity. Besides, there are many studies about fast access techniques in HBase, where indexing is mostly used. The Reference [10,11] reviews the current indexing techniques in HBase and compares the characteristics and applications of several indexing techniques. In the Reference [12,13], the Z curve is used to reduce the dimensionality of the spatial data; the spatial information and text information of the spatial text object are simultaneously indexed through an effective data allocation strategy [14,15]. References [16,17] are the research on big data storage and retrieval schemes based on distribution NoSQL database. The Reference [18,19] propose a storage optimization method for vector spatial data from the relational database to the distributed NoSQL database. The storage and retrieval method through associated multi-attributes of massive data is described in Reference [20], which solves the secondary index problem based on the multi-condition query of HBase dynamic properties [21]. Although the indexing technique can improve retrieval speed significantly, the disadvantage is that the maintenance is relatively complex. There are also some researches that put forward the corresponding optimization solution for distributed database storage and retrieval based on the characteristics of data storage of the specific application domain, which have achieved relatively satisfactory results [22,23]. However, the versatility is weak because the optimization aims at the specific application environment.

The studies in the above references are generally directed at a specific area of use. In this paper, aiming at the problem mentioned above, we study converting and storing the data file stored by row to HBase distributed database, and fast retrieval and access to the big data in HBase. It involves the storage domain of big data. A common tool for distributed storage, transformation and retrieval of big data is to be studied.

2 The Specific Issue to Solve of This Research

Distributed No-SQL databases are gradually being used in more and more systems due to their large capacity, easy expansion and fast retrieval speed [24,25]. However, the problem that needs to be solved in practical application is how to convert and store the existing massive data into a non-SQL database, and the converted data must be retrieved quickly.

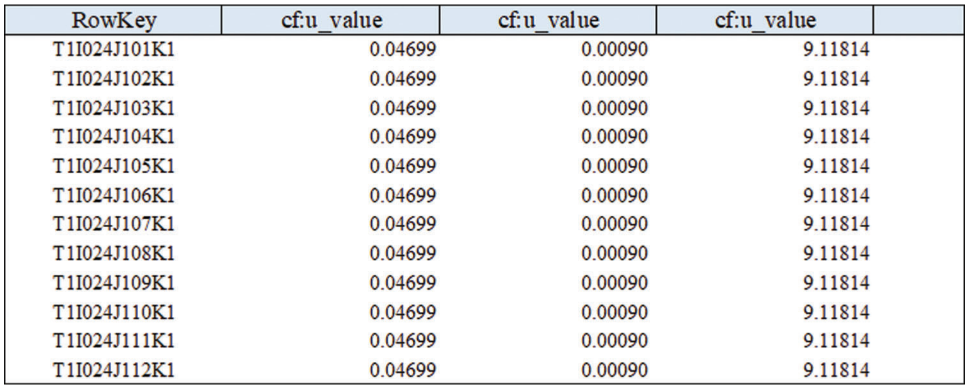
Concretely, data files are defined differently. For example, in a row of data, some attributes are for marking and distinguishing, such as time, coordinates, numbers, etc., while others are descriptive text. As shown in Fig. 1, the first two fields of the data file represent attribute information for coordinates x and y, while the remaining fields are descriptive information about specific parameter values. Due to the different structures, field names and types of source data files, the storage mode of data items is also different, so it is difficult to design a general method to import data using the current HBase import tool such as "bulkloader". Generally, to solve this kind of problem, programmers should write a specific program to extract and import the data. However, this program only applies to the specific source data file or the destination table. If they have changed, the program would be no longer applicable. Therefore, the general conversion of file data stored in rows to the HBase distributed database is needed.



```
  TITLE="QSH T=  4.9999999E-03 d"
 VARIABLES="X ","Y ","U1 ","V1 ","Z1 ","U2 ","V2 ","Z2 ","U3 ","V3 ","Z3 ","U4 ","V4 ","Z4 ","U5 ","V5 ","Z5 "
  ZONE N=       24910  E=        24414  F=FEPOINT,ET=QUADRILATERAL
14        137       0.00090       9.11814       0.00000        0.00000       0.00000       0.00000
14        138       0.00090       9.11814       0.00000       -0.00097       0.00000       0.00000
14        193       0.00090       9.11814       0.00000        0.00000       0.00000       0.00000
14        194       0.00090       9.11814       0.00000       -0.00092       0.00000       0.00000
15        136       0.00000       0.00000       0.00000        0.00000       0.00000       0.00000
15        137       0.00090       9.11814       0.00111       -0.00117       0.00000      -0.00020
15        138       0.00090       9.11814       0.00098       -0.00102       0.00000      -0.00033
15        139       0.00090       9.11814       0.00000       -0.00108       0.00000       0.00000
15        140       0.00090       9.11814       0.00000       -0.00104       0.00000       0.00000
15        193       0.00090       9.11814       0.00107        0.00000       0.00000      -0.00019
15        194       0.00090       9.11814       0.00094       -0.00096       0.00000      -0.00032
15        195       0.00090       9.11814       0.00000       -0.00103       0.00000       0.00000
15        196       0.00090       9.11814       0.00000       -0.00100       0.00000       0.00000
```

Figure 1: The example of the file

This paper, based on HBase, introduces the feature-based conversion of big data files into distributed No-SQL database and fast retrieval technology. Retrieval needs to return data matching the specified attribute information, which is called the feature [26–28]. The data in HBase is accessed by “RowKey”, shown in Fig. 2. To achieve this, the HBase row key “RowKey” in the specified destination table should be generated automatically according to the data retrieval requirements in different files. At the same time, it is necessary to ensure that the “RowKey” of each row in HBase is unique and can be retrieved and accessed quickly.



RowKey	cf:u_value	cf:u_value	cf:u_value	
T1I024J101K1	0.04699	0.00090	9.11814	
T1I024J102K1	0.04699	0.00090	9.11814	
T1I024J103K1	0.04699	0.00090	9.11814	
T1I024J104K1	0.04699	0.00090	9.11814	
T1I024J105K1	0.04699	0.00090	9.11814	
T1I024J106K1	0.04699	0.00090	9.11814	
T1I024J107K1	0.04699	0.00090	9.11814	
T1I024J108K1	0.04699	0.00090	9.11814	
T1I024J109K1	0.04699	0.00090	9.11814	
T1I024J110K1	0.04699	0.00090	9.11814	
T1I024J111K1	0.04699	0.00090	9.11814	
T1I024J112K1	0.04699	0.00090	9.11814	

Figure 2: The data structure after conversion

Aiming at the problem, we propose a general HBase data conversion storage technique based on mapping models. “HBase Thrift” client is used to generate row keys through a defined “RowKey” expression and import the data stored in rows into the HBase database. On the premise of consistency, according to a variety of combinations, the multiple features of data objects are formed into row keys which are stored in multiple HBase data tables with column data redundantly. In the multi-feature data retrieval, by matching several feature values in the row key, a fuzzy result set can be quickly obtained, and the fuzzy result set can be further filtered to obtain the final accurate result set. Our proposed method could be applied to other kinds of files due to the user-defined import rules and mapping tables. The row key storage data formed according to the multi-feature combination mode can provide a fast data retrieval interface.

3 Technical Scheme

In order to convert a row format file into an HBase file, we designed a mapping model that shows the mapping relationship between the source file field and the target HBase table, where the “RowKey” generation expression describes the mapping file from the source file to HBase.

The “RowKey” is unique and automatically generated by extracting the required characters from the file name, the field name and the field value, where we may truncate and reformat these characters if necessary. Besides, to speed up retrieval and access, we adopt the prefix matching on the feature of “RowKey” and set the matching priority.

3.1 The HBase Data Conversion Based on the Mapping Model

The mapping model describes the convertible file objects, the mapping from file object fields to HBase columns, and the specific conversion scheme. The model consists of three kinds of data structures: “DataFileObject”, “DataFileFields” and “ImportScheme”.

The data file object is the mapping source of HBase big data, and its data structure shows the essential attribute of a kind of data file object:

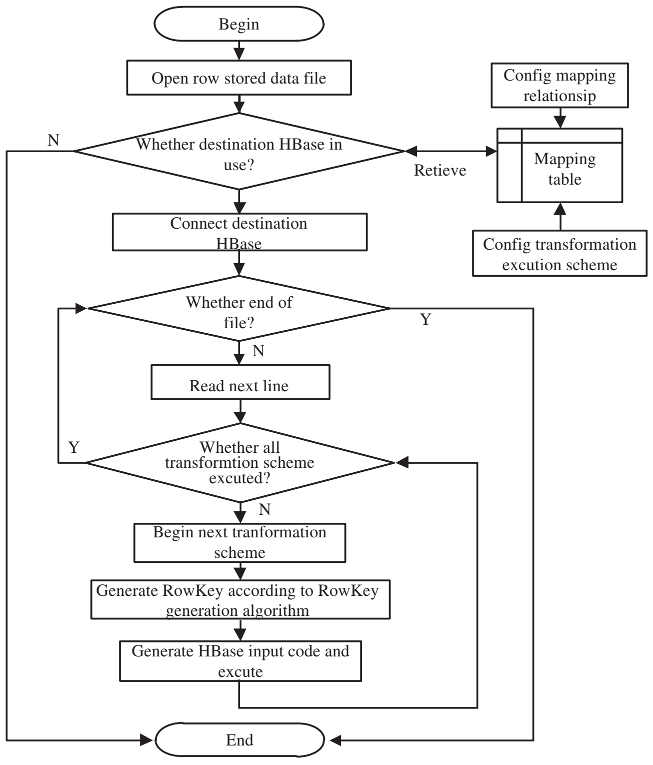

Begin
Open row stored data file
Config mapping relationsip
Whether destination HBase in use?
N
Retieve
Mapping table
Connect destination HBase
Config transformation excution scheme
Whether end of file?
Y
N
Read next line
Whether all transformtion scheme excuted?
Y
N
Begin next tranformation scheme
Generate RowKey according to RowKey generation algorithm
Generate HBase input code and excute
End

Figure 3: Flow chart of conversion

Structure of DataFileObject

{

String MODEL_ID; // The identifier of data file, uniqueness and non-null

String MODEL_NAME; // The name of data file

Integer FIELD_COUNT; // The number of fields

String MODEL_MEMO; // The description of data file

}

The mapping structure is a detailed description of the data file object fields. Each instance of the data structure describes one kind of field mapping relationship. A data file object can contain multiple field mapping relationships. The data structure which describes the mapping from data fields to HBase columns is as follows:

Structure DataFileFields

{

String FIELD_ID; //The identifier of field, uniqueness and non-null

String MODEL_ID; // The identifier of data file object of field

String FIELD _NAME; //The name of field

String COLUMN_FAMILY; //The field corresponding name of column cluster in HBase

String COLUMN_QUALIFIER; // The column qualifier in HBase of field corresponding

Integer FIELD _INDEX; // The position index in every row of field in data file object

```
String FIELD _DATATYPE; // The data type of field
}
```

The conversion scheme defines the details of data conversion and storage. The mapping model of data structure which describes the importing execution plan is as follows:

```
Structure ImportScheme
{
String EXPRINCIPLE_ID; //The identifier of import scheme, uniqueness and non-null
String EXPRINCIPLE_NAME; //The name of import scheme
String MODEL_ID; //The identifier of data file object which import scheme correspond to
String HOST; // The server of objective HBase
String TABLE_NAME; // The name of table of objective HBase
Integer PORT_NUM; // The port number of server of objective HBase
String ROWKEY_EXPRESSION; // The expression of row key which to be generated
String FIELD_COLLECTION; // The field collection of data file to be imported
}
```

In order to improve the execution efficiency, a conversion cycle starts from opening a data file object. One cycle could execute several different execution schemes, which splits a row of data into multiple HBase data rows or imports the data into different HBase instances according to different mapping rules.

The mapping relationship described by the "DataFileFields" data structure and the conversion scheme described by the "ImportScheme" data structure are associated with data item MODEL_ID to the data file object defined by the "DataFileObject" data structure. Because a data file can contain more than one column mapping relation and a single execution can correspond to multiple storage solutions at the same time, it is a one-to-many relationship between the data structure defined by "DataFileObject" and the data structures defined by "DataFileFields" and "ImportScheme".

As shown in Fig. 3, the converting of a source file into HBase data based on the mapping model is as follows:

1) Begin;
2) Open the data file stored in row;
3) Check whether the objective HBase is available, if it is not available, go to 11), otherwise, go on;
4) Connect objective HBase;
5) Judge whether the reading file has reached the end, if it ends, go to 11), otherwise go on;
6) Read the next row data of the source file;
7) Judge whether all the import schemes have finished, if not go to 5), otherwise go on;
8) Begin to execute next import scheme;
9) Generate row key on the generating algorithm of "RowKey";
10) Generate the execute code to import HBase, and go to 7) after the execution has been finished.
11) End.

The process firstly sets up the field mapping model of the data file and imports the conversion scheme; secondly opens the data file to check its availability; finally reads the data file by row, traverses all the import schemes that need to be executed, generates row keys according to the "RowKey" generation rules of each scheme, then writes the data into the corresponding columns of the destination table according to the mapping model. After all the rows in the data file are executed, the import process ends.

3.2 The Expression of "Rowkey" and the Generating Algorithm of "Rowkey"

The "RowKey" expression is a set of simplified encoding and decoding rules. The "RowKey" in HBase contains feature values. The "RowKey" is generated by the following steps. We first abstract and determine the source of the "RowKey" and then define the rule of the "RowKey" expression. When data are converting, the algorithm automatically extracts multiple feature values from the specified location, and they are combined into a "RowKey" after some conversion. Specifically, each "RowKey" imported into HBase is generated by calculating the "RowKey" expression on the fixed character, the field name and the field name value.

"RowKey" expression definition rules are as follows:

1) Fixed Character: The expression appears in row key in the form of fixed character, for example, "ABC";
2) Escape Character: The expression is as "[]", except for the fixed character, all content need to appear in the escape character. When decoding, the content in the brackets is escaped calculation;
3) File Name: The expression is as "F", which means the file name of the acquired data file, such as "TEC_AA_L01_00001.DAT";
4) Field Name: The expression is as "C (index)", which means to acquire the field name whose index is "index" in the field collection. For example, "[C(0)]" means to get the field name "MODEL_ID" ;
5) Field Value: The expression is as "V (name index)", which means to obtain the field value from the field name or index number. The field name needs to be quoted in double quotes. If using an index number, the value must be an integer. For example, "[V(0)]" means to acquire the field value "MDS_Z0120150217001";
6) Truncation Operator: The expression is as ".S (start, end)", which means to get the characters from "start" position to "end" position. For example, "[V(0). S(7, 10)]" means to acquire the result "2015";
7) Formation String: The expression is as ".T (format)", it means to transform the string to the "format" format. For example, "["12".T("000")]"means to acquire the result "012";
8) Mathematical Operators: The expression is as "+" or "-", which is a mathematical operator. For example, "[V (0) .S (7, 10)]-1" means to acquire the result "2014";

Based on the above definitions of row key expressions, the row key generation algorithm is as follows:

1) Read row key expression string and decompose the string to the collection of characters which is expressed as "EXP";
2) Initialize variables: Result string SR="", state variable ST="", escape control character TB=false, temporary cache strings TMP="", loop control variable $i = 0$. Then traverse row key expression character set EXP;
    a) Judging escape character "TB", if it is "FALSE" then it expresses the state of non-escape and go to b) or else it is "TRUE" which expresses the state of escape and go to c);
    b) Judge whether "EXP[i]" is equal to '[', if it is, it means the escape begins, and "TB" is set to "TRUE", otherwise, "EXP[i]" is added to the result string and go to h);
    c) Judge whether "EXP[i]" is equal to ']', if it is, it means the escape ends, and "TB" is set to "FALSE", and the string TMP is added to the result string SR and go to h), otherwise, go to d);
    d) Judge the string control character SB, if it is "TRUE", it means it is in the character state, if EXP[i] is "", it means the character state is over, TB is set to "FALSE", otherwise, EXP[i] is added to the temporary buffer string TMP, and then turn to h), if SB is "FALSE", turn to e) or else continue;
    e) Judging the string "EXP[i]", if it is "" then it expresses the state of beginning and "TB" is set as "TRUE" then go to h); if the string "EXP[i]" is a character 'F' then the file name is put into temporary string TMP or else if it is a character 'C' or 'V' or 'S' or 'T' then set the state variable ST as corresponding character and go to h) or else go to f);

f) When the string “EXP[i]” is “)”, it expresses ending character with parameters. If the character type in state variable ST is ‘C’ then the column name is put into TMP. If is ‘V’ then the column value is put into TMP. If is ‘S’ then temporary string TMP is truncated in parameter. And if is ‘T’ then temporary string TMP is formatted by calling the method of “String.to String (format)” in the parameter format. Then go to h) after ending;
g) If the string “EXP[i]” cannot be judged, go to h);
h) I = i+1, if the “i” less or equal to the largest row number of collect R then go to a) or else go to 3);.
i) Return the result string SR.

For example, if the expression of “RowKey” is ["T" F.S (9, 2) "K" V(1). T("000")], the generated rule is to take the fixed character "t", keep two characters from the ninth character of the file name, take the fixed character K, take the value of the first column and convert it into the numerical lattice “000”, and finally group these characters in order. The data file generates row keys according to “RowKey” expressions and stores them into HBase, where the conversion process is as Fig. 4.

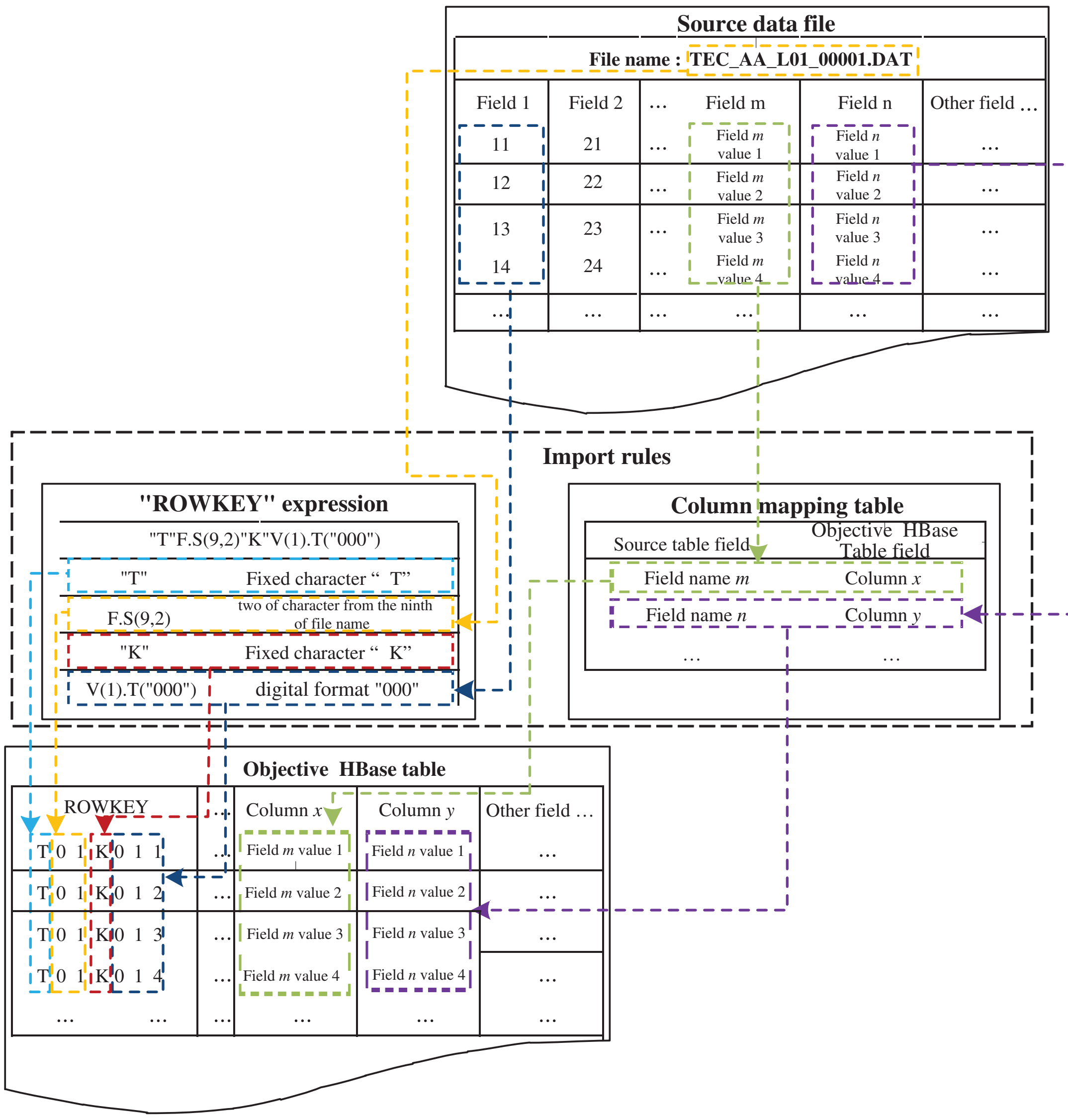


Figure 4: The storage conversion process of data file

In the above generate rules, the “RowKey” of the first row (in the destination HBase table of Fig. 4) is “T01K011”, the “RowKey” of the second row is “T01K012”, the “RowKey” of the third row is “T01K013”, the “RowKey” of the fourth row is “T01K014”, the column m and column n in the source file correspond to the column x and the column y in the table of HBase respectively through mapping model. The conversion of other columns is similar.

The traditional retrieval method is data column matching query [29], which requires a full table scan, so it is inefficient. We store the HBase data row in the order of “RowKey”, and retrieve data through “RowKey” feature matching, which can achieve a faster response speed [30,31].

In addition, the longest prefix matching of “RowKey” should be satisfied, that is, the feature value should match the first few characters of “RowKey” as much as possible. The fastest retrieval speed can be obtained by calling the “ScanWithPrefix” interface in HBase. Therefore, we put the common feature attribute at the head position of the “RowKey”, or use multiple “RowKey” rules to retrieve multiple feature attributes. Besides, the way of redundant storage of data is adopted to achieve fast retrieval of HBase big data based on feature values. Specific solutions are as follows.

3.2.1 Multiple Feature Value Index Storage Schemes for a Large Amount of Data

According to the characteristics of HBase, we optimize data storage. On the premise of consistency, according to the needs of retrieval, the “RowKey” is composed of multiple attributes of data objects and is stored in one or more HBase data tables together with common column values redundantly. The fuzzy result set is obtained fast by matching several feature values of row key when the multi-feature data are retrieved. If the feature-based retrieval cannot meet the requirements, the way of filtering the fuzzy result set is taken to obtain the final result set.

3.2.2 The Rule to Generate Multi Feature “Rowkey”

The data are stored in HBase by the “RowKey” dictionary order. To improve the efficiency of multi-keyword retrieval, we propose a method of generating “RowKey” according to the feature, which puts the frequently-used features in the head position of “RowKey”. For example, a series of index data representing 3D space of multiple attribute datasets is shown in Tab. 1. If the feature 3 is required as the retrieval condition, that is, a certain data crosscut with fixed z-coordinate is retrieved; the finally generated “RowKey” and data storage form in HBase are shown in Tab. 2. If the feature 1 is required, the final results are shown in Tab. 3.

Table 1: The attribute data set of multi-feature

Feature				Data column			
Feature 1	Feature 2	Feature 3	…	Column a	Column b	Column c	…
x1	y1	z1	…	a1	b1	c1	…
x1	y2	z2	…	a2	b2	c2	…
x2	y3	z2	…	a3	b3	c3	…
…	…	…	…	…	…	…	…

In this way, the data with similar feature values can be stored as closely as possible, and the “RowKey” prefix matching can be used to filter conveniently. The results are filtered to get the final result set, which avoids frequent full table scanning when retrieving data and improves the retrieval efficiency. Besides, to meet the needs of fast retrieval, a number of import rules are created for a certain type of data files. When the “RowKey” expression is set, a sequence of feature values extracted is ordered according to the retrieval need. The "RowKey" and the description data which are both obtained in different orders of feature values are stored in multiple tables in the way of redundant storage.

Table 2: Structure of HBase (feature 3 is used as retrieval condition)

RowKey	Column family: cf			
	Column a	Column b	Column c	…
z1\$x1\$y1	a1	b1	c1	…
z2\$x1\$y2	a2	b2	c2	…
z2\$x2\$y3	a3	b3	c3	…
…	…	…	…	…

Table 3: Structure of HBase (feature 1 is used as retrieval condition)

RowKey	Column family: cf			
	Column a	Column b	Column c	…
x1\$z1\$ y1	a1	b1	c1	…
x1\$z2\$ y2	a2	b2	c2	…
x2\$z2\$ y3	a3	b3	c3	…
…	…	…	…	…

3.2.3 The Fast Retrieve and Access Method

There are two ways to retrieve and access converted HBase data. According to the requirements of cross-language and cross-platform data access, the HBase cluster needs to open the "Thrift" service and provides a data access interface supporting multiple languages for "HBase Thrift" clients [32].

a) The Prefix Matching Retrieve

Since HBase stores data in the dictionary order of "RowKey", the most effective method is to search through the "RowKey" prefix matching. In the open "HClient" instance which specifies the target table, we could directly call the "ScanWithPrefix" method to get the result set.

b) The Feature Matching Retrieve

According to the previous multi-feature value "RowKey" generation rules, for feature-based retrieval, the common-used features are placed in the front position of "RowKey" when one is designing the HBase table structure. Therefore, the tables whose feature values are in the head positions appear at the top of the query results. If using multi-feature query, we could call the "Scan" method and set "rowfilter" for data retrieval.

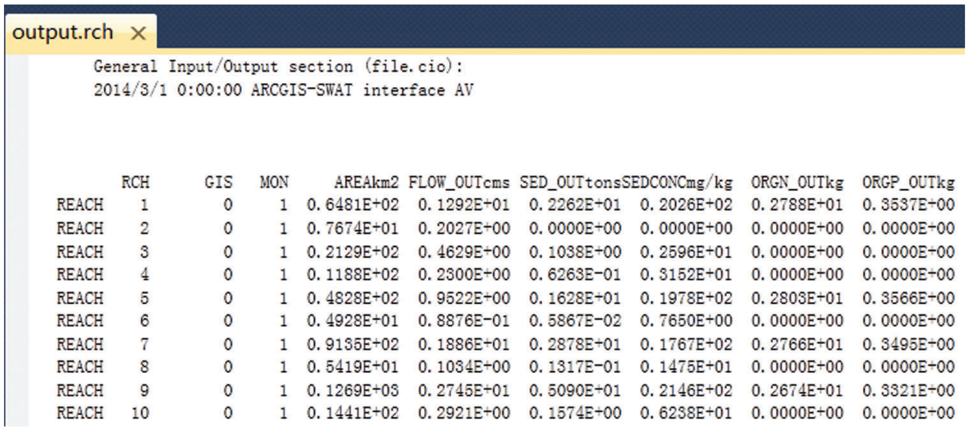
4 Test and Analysis of Big Data Retrieval

This research is derived from the model integration system of "The integration technology research of three gorges reservoir area and upstream basin water environment risk assessment and pre-warning platform" which is the sub-project of "The Water Environment Risk Assessment and Pre-warning Technology Research and Demonstration of the Three Gorges Reservoir Area and Upstream Basin". We should save the output to visualize the output of the model. However, the amount of data generated by a day is up to tens of GB. Therefore aiming at the problem of a large amount of computation, we adopt the big data idea, which uses the HDFS distributed storage cluster to store massive data and provides a stable, efficient, and scalable storage scheme [33]. The efficient indexing technology of model data based on HBase has provided high-speed data retrieval services for the real-time demonstration of model, historical event playback of the model, model analysis and other business.

4.1 Conversion in Different Types of Data File

In order to verify the validity of data transformation, two different types of model data files were selected. Hadoop platform consists of 8 virtual machines, including 1 set as name-node and *h* master and provides thrift interface, the other 7 as data-node and region server [34]. The virtual machines are configured for dual core 2.0G Hz CPU, 2GB memory and 1TB hard disk, connected by gigabit network cards. Two different types of model data files were recorded as model file A and model file B.

As shown in Fig. 5, summary information such as time and version are stored in the header part of model file A, feature values are stored in the first three columns at the main part of the data file, and the common values are stored in the last 12 columns. To meet the need for data conversion, “RowKey” in HBase after conversion is composed of time string and feature values in the first three columns that were formatted. Therefore, the “RowKey” expression to convert is ["201403010000RCH"V(1). T("000")"GIS" V(2). T("000")"MON"V(3). T("000”)]. Then data in the last 12 columns is stored in the corresponding column under column family in HBase.



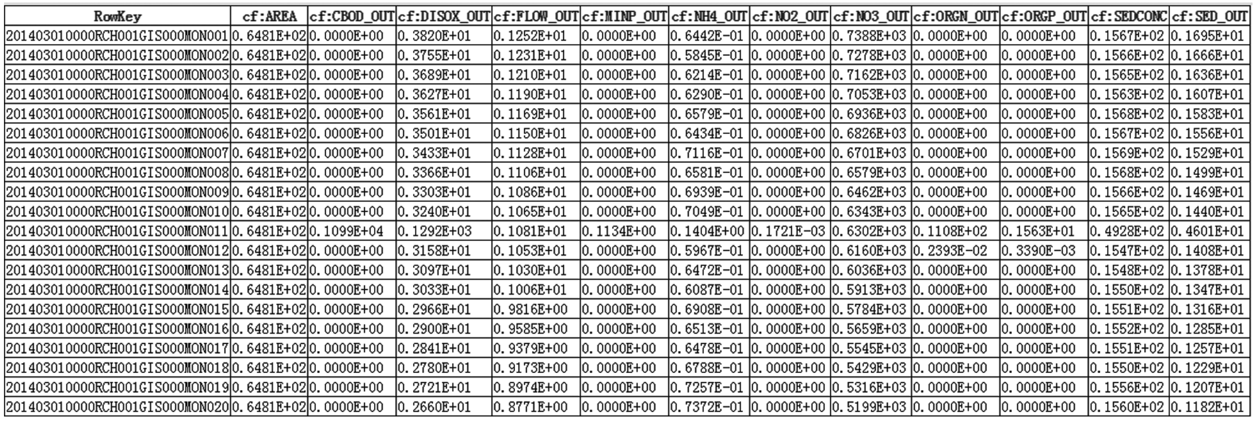
```
General Input/Output section (file.cio):
2014/3/1 0:00:00 ARCGIS-SWAT interface AV



      RCH      GIS   MON    AREAkm2 FLOW_OUTcms SED_OUTtonsSEDCONCmg/kg ORGN_OUTkg  ORGP_OUTkg
REACH   1        0     1  0.6481E+02  0.1292E+01  0.2262E+01  0.2026E+02  0.2788E+01  0.3537E+00
REACH   2        0     1  0.7674E+01  0.2027E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
REACH   3        0     1  0.2129E+02  0.4629E+00  0.1038E+00  0.2596E+01  0.0000E+00  0.0000E+00
REACH   4        0     1  0.1188E+02  0.2300E+00  0.6263E-01  0.3152E+01  0.0000E+00  0.0000E+00
REACH   5        0     1  0.4828E+02  0.9522E+00  0.1628E+01  0.1978E+02  0.2803E+01  0.3566E+00
REACH   6        0     1  0.4928E+01  0.8876E-01  0.5867E-02  0.7650E+00  0.0000E+00  0.0000E+00
REACH   7        0     1  0.9135E+02  0.1886E+01  0.2878E+01  0.1767E+02  0.2766E+01  0.3495E+00
REACH   8        0     1  0.5419E+01  0.1034E+00  0.1317E-01  0.1475E+01  0.0000E+00  0.0000E+00
REACH   9        0     1  0.1269E+03  0.2745E+01  0.5090E+01  0.2146E+02  0.2674E+01  0.3321E+00
REACH  10        0     1  0.1441E+02  0.2921E+00  0.1574E+00  0.6238E+01  0.0000E+00  0.0000E+00
```

Figure 5: Data structure of model file A

Data in HBase after conversion are shown in Fig. 6.

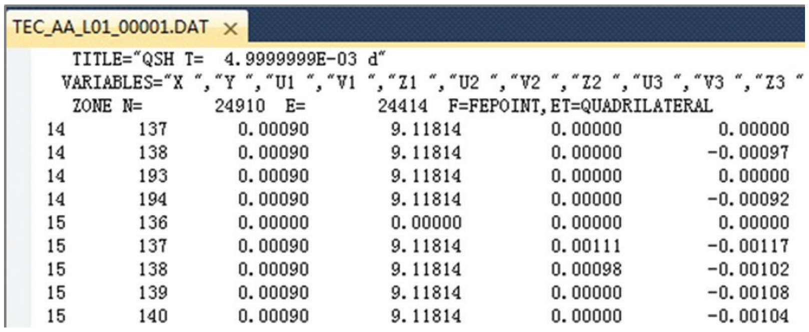
As shown in Fig. 7, summary information such as time and version are stored in the header part of model file A, part of feature values are stored in the first two columns at the main part of the data file, and the common values are stored in the last nine columns. Three categories of value denoted as U, V and Z are stored in the last nine columns. For example, U1 is composed of category name U and a feature value 1.

One data-row in the file is transformed into three records in the HBase, so three different conversion plans are required to complete this job. In the first conversion plan, the “RowKey” expression is ["T"F.S (11,5)"I"V(43). T("000")"J"V(44). T("000")"K1"], and the columns of common values are U1 and V1 and Z1. In the second conversion plan, the “RowKey” expression is ["T"F.S(11, 5)"I"V(43). T ("000")"J"V(44). T("000")"K2"], and the columns of common values are U2 and V2 and Z2. In the last conversion plan, the “RowKey” expression is ["T"F.S(11,5)"I"V(43). T("000")"J"V(44). T("000")"K3"], and the columns of common values are U3 and V3 and Z3.



RowKey	cf:AREA	cf:CBOD_OUT	cf:DISOX_OUT	cf:FLOW_OUT	cf:MINP_OUT	cf:NH4_OUT	cf:NO2_OUT	cf:NO3_OUT	cf:ORGN_OUT	cf:ORGP_OUT	cf:SEDCONC	cf:SED_OUT
201403010000RCH001GIS000MON001	0.6481E+02	0.0000E+00	0.3820E+01	0.1252E+01	0.0000E+00	0.6442E-01	0.0000E+00	0.7388E+03	0.0000E+00	0.0000E+00	0.1567E+02	0.1695E+01
201403010000RCH001GIS000MON002	0.6481E+02	0.0000E+00	0.3755E+01	0.1231E+01	0.0000E+00	0.5845E-01	0.0000E+00	0.7278E+03	0.0000E+00	0.0000E+00	0.1566E+02	0.1666E+01
201403010000RCH001GIS000MON003	0.6481E+02	0.0000E+00	0.3689E+01	0.1210E+01	0.0000E+00	0.6214E-01	0.0000E+00	0.7162E+03	0.0000E+00	0.0000E+00	0.1565E+02	0.1636E+01
201403010000RCH001GIS000MON004	0.6481E+02	0.0000E+00	0.3627E+01	0.1190E+01	0.0000E+00	0.6290E-01	0.0000E+00	0.7053E+03	0.0000E+00	0.0000E+00	0.1563E+02	0.1607E+01
201403010000RCH001GIS000MON005	0.6481E+02	0.0000E+00	0.3561E+01	0.1169E+01	0.0000E+00	0.6579E-01	0.0000E+00	0.6936E+03	0.0000E+00	0.0000E+00	0.1568E+02	0.1583E+01
201403010000RCH001GIS000MON006	0.6481E+02	0.0000E+00	0.3501E+01	0.1150E+01	0.0000E+00	0.6434E-01	0.0000E+00	0.6826E+03	0.0000E+00	0.0000E+00	0.1567E+02	0.1556E+01
201403010000RCH001GIS000MON007	0.6481E+02	0.0000E+00	0.3433E+01	0.1128E+01	0.0000E+00	0.7116E-01	0.0000E+00	0.6701E+03	0.0000E+00	0.0000E+00	0.1569E+02	0.1529E+01
201403010000RCH001GIS000MON008	0.6481E+02	0.0000E+00	0.3366E+01	0.1106E+01	0.0000E+00	0.6581E-01	0.0000E+00	0.6579E+03	0.0000E+00	0.0000E+00	0.1568E+02	0.1499E+01
201403010000RCH001GIS000MON009	0.6481E+02	0.0000E+00	0.3303E+01	0.1086E+01	0.0000E+00	0.6939E-01	0.0000E+00	0.6462E+03	0.0000E+00	0.0000E+00	0.1566E+02	0.1469E+01
201403010000RCH001GIS000MON010	0.6481E+02	0.0000E+00	0.3240E+01	0.1065E+01	0.0000E+00	0.7049E-01	0.0000E+00	0.6343E+03	0.0000E+00	0.0000E+00	0.1565E+02	0.1440E+01
201403010000RCH001GIS000MON011	0.6481E+02	0.1099E+04	0.1292E+03	0.1081E+01	0.1134E+00	0.1404E+00	0.1721E-03	0.6302E+03	0.1108E+02	0.1563E+01	0.4928E+02	0.4601E+01
201403010000RCH001GIS000MON012	0.6481E+02	0.0000E+00	0.3158E+01	0.1053E+01	0.0000E+00	0.5967E-01	0.0000E+00	0.6160E+03	0.2393E-02	0.3390E-03	0.1547E+02	0.1408E+01
201403010000RCH001GIS000MON013	0.6481E+02	0.0000E+00	0.3097E+01	0.1030E+01	0.0000E+00	0.6472E-01	0.0000E+00	0.6036E+03	0.0000E+00	0.0000E+00	0.1548E+02	0.1378E+01
201403010000RCH001GIS000MON014	0.6481E+02	0.0000E+00	0.3033E+01	0.1006E+01	0.0000E+00	0.6087E-01	0.0000E+00	0.5913E+03	0.0000E+00	0.0000E+00	0.1550E+02	0.1347E+01
201403010000RCH001GIS000MON015	0.6481E+02	0.0000E+00	0.2966E+01	0.9816E+00	0.0000E+00	0.6908E-01	0.0000E+00	0.5784E+03	0.0000E+00	0.0000E+00	0.1551E+02	0.1316E+01
201403010000RCH001GIS000MON016	0.6481E+02	0.0000E+00	0.2900E+01	0.9585E+00	0.0000E+00	0.6513E-01	0.0000E+00	0.5659E+03	0.0000E+00	0.0000E+00	0.1552E+02	0.1285E+01
201403010000RCH001GIS000MON017	0.6481E+02	0.0000E+00	0.2841E+01	0.9379E+00	0.0000E+00	0.6478E-01	0.0000E+00	0.5545E+03	0.0000E+00	0.0000E+00	0.1551E+02	0.1257E+01
201403010000RCH001GIS000MON018	0.6481E+02	0.0000E+00	0.2780E+01	0.9173E+00	0.0000E+00	0.6788E-01	0.0000E+00	0.5429E+03	0.0000E+00	0.0000E+00	0.1550E+02	0.1229E+01
201403010000RCH001GIS000MON019	0.6481E+02	0.0000E+00	0.2721E+01	0.8974E+00	0.0000E+00	0.7257E-01	0.0000E+00	0.5316E+03	0.0000E+00	0.0000E+00	0.1556E+02	0.1207E+01
201403010000RCH001GIS000MON020	0.6481E+02	0.0000E+00	0.2660E+01	0.8771E+00	0.0000E+00	0.7372E-01	0.0000E+00	0.5199E+03	0.0000E+00	0.0000E+00	0.1560E+02	0.1182E+01

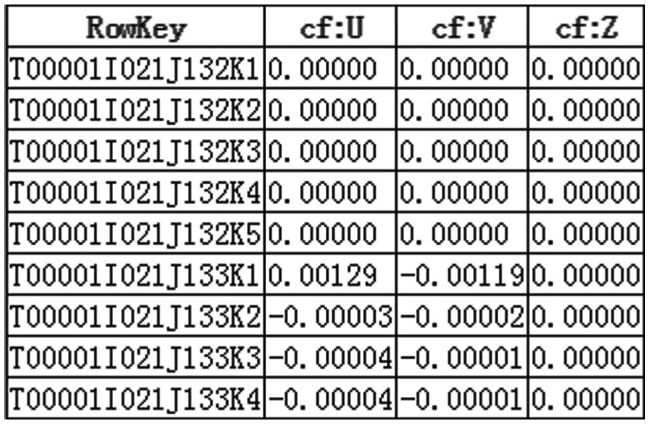
Figure 6: Data structure of model file A in HBase after conversion



```
TEC_AA_L01_00001.DAT
   TITLE="QSH T=  4.9999999E-03 d"
  VARIABLES="X ","Y ","U1 ","V1 ","Z1 ","U2 ","V2 ","Z2 ","U3 ","V3 ","Z3 "
   ZONE N=       24910   E=       24414   F=FEPOINT,ET=QUADRILATERAL
14        137        0.00090         9.11814          0.00000          0.00000
14        138        0.00090         9.11814          0.00000         -0.00097
14        193        0.00090         9.11814          0.00000          0.00000
14        194        0.00090         9.11814          0.00000         -0.00092
15        136        0.00000         0.00000          0.00000          0.00000
15        137        0.00090         9.11814          0.00111         -0.00117
15        138        0.00090         9.11814          0.00098         -0.00102
15        139        0.00090         9.11814          0.00000         -0.00108
15        140        0.00090         9.11814          0.00000         -0.00104
```

Figure 7: Data structure of model file B

Data in HBase after conversion are shown in Fig. 8.



RowKey	cf:U	cf:V	cf:Z
T00001I021J132K1	0.00000	0.00000	0.00000
T00001I021J132K2	0.00000	0.00000	0.00000
T00001I021J132K3	0.00000	0.00000	0.00000
T00001I021J132K4	0.00000	0.00000	0.00000
T00001I021J132K5	0.00000	0.00000	0.00000
T00001I021J133K1	0.00129	-0.00119	0.00000
T00001I021J133K2	-0.00003	-0.00002	0.00000
T00001I021J133K3	-0.00004	-0.00001	0.00000
T00001I021J133K4	-0.00004	-0.00001	0.00000

Figure 8: Data structure of model file B in HBase after conversion

4.2 Fast Retrieve Based on "Rowkey"

To meet the need for fast retrieval in HBase after conversion, "RowKey" is formed in different combinations of features, and data redundancy is stored in different HBase tables. We use data in model file B to conduct experiments. The "RowKey" expression in conversion plan is ["T"F.S(11, 5)"I"V(43).

T("000") "J"V(44). T("000")"K1"], which have put the filename index in the head position of the “RowKey”. Therefore, data files transformed in this conversion plan could be retrieved fast through the filename index.

The actual data in the project is converted into HBase No-SQL database through the conversion plan, and the total number of records exceeds 50 million. We use the same retrieve parameters to compare the “Scan With Prefix” retrieval and the common retrieve which is using the feature value matching method [35–37]. After 30 times of retrieval experiments, the response time of each retrieve parameter is as shown in Fig. 9.

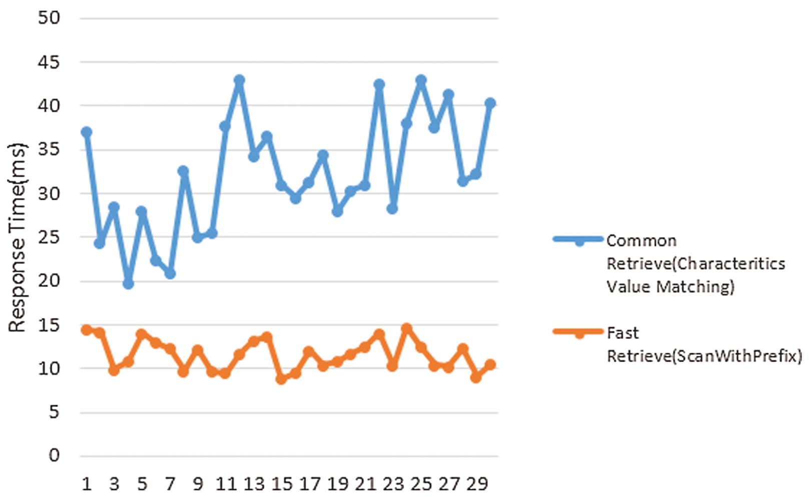


Figure 9: Comparison of common retrieve and fast retrieve

Then, in order to test the fast retrieval through coordinates, a new conversion plan with a “RowKey” expression ["I"V(43).T("000")"J"V(44).T("000")"K1""T"F.S(11,5)] is created. Similarly, we compare the method of using the “Scan With Prefix” and the fast retrieval of common feature value. Because the coordinates have many parameters, it takes a long time to complete the retrieval of multiple features. The results are shown in Fig. 10.

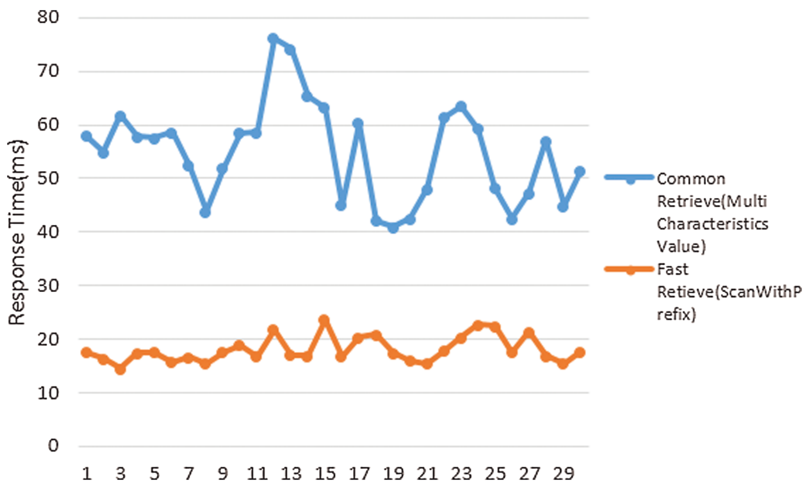


Figure 10: Comparison of multi feature value common retrieve and fast retrieve

5 Conclusion

The research results of this paper can be applied to most cases of big data files from row storage to HBase single line or multi-line storage, as well as to quickly retrieve and access data in different application environments. This research has been applied to the data storage and quick retrieval of the following three national scientific and technological innovation projects including “Three Gorges Reservoir area and upstream basin”, “Research on GIS Voice Query Based on Natural Language Understanding” and “Cloud Storage Based Smart City Big Data Processing and Key Technology Research”. The application is of great help to improve the efficiency of video data access. It has also been adopted in the sub-project data storage and retrieval of model integration system of “Three Gorges Reservoir Area and Upstream Basin Water Environment Risk Assessment and Early Warning Technology Research and Demonstration”. Although the data formats and types of the two applications are different, this method is also applied in the above mentioned two fields with great storage efficiency and retrieval effect, which improves its general applicability.

In the future, this method will be extensively adopted in distributed No-SQL database big data storage and retrieval with broad prospects. It also can be developed into a commercial level big data storage and high-speed retrieval management tool.

Acknowledgement: I want to dedicate this work to my friends Yunhua Wang, Shizi Wang, and Longhua Xue. Further, I am very grateful to the reviewers and Journal Authorities.

Funding Statement: This work was sponsored by the Natural Science Research Program of Higher Education, Jiangsu Province (19KJD520005) and Qing Lan Project of Jiangsu Province(Su Teacher's Letter [2021] No. 11).

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] H. Li, L. Zhu, G. Sun and L. Wang, “Design and implementation of distributed mass small file storage system,” *Computer Engineering and Design*, vol. 37, no. 1, pp. 86–92, 2016.

[2] J. B. Karanjekar and M. B. Chandak, “Uniform query framework for relational and NoSQL databases,” *Computer Modeling in Engineering & Sciences*, vol. 113, no. 2, pp. 171–187, 2017.

[3] Q. Zhang, Y. Chen, C. P. Zhang and M. Liu, “Overview of unstructured power data storage base on distributed file system,” *Computing Application System*, vol. 26, no. 2, pp. 30–36, 2017.

[4] C. Piao, “The distributed storage method of large-scale unstructured data,” *Journal of Taiyuan Normal University*, vol. 20, no. 3, pp. 57–61, 2021.

[5] J. Sun, “*Research on Data Processing Technology Based on HBase*,” M.S. thesis, Dept. Computer, People's Public Security University of China, Beijing, China, 2019.

[6] K. Shao, W. Jiang and J. Lv, “Study and realization of a fast parallel import tool for very-large data,” *Computer Applications and Software*, vol. 32, no. 9, pp. 26–30, 2015.

[7] Y. S. Wijaya and A. A. Arman, “A framework for data migration between different datastore of NoSQL database,” in *Proc. Int. Conf. on ICT for Smart Society 2018 (ICISS 2018)*, West Java, Indonesia, pp. 1–6, 2018.

[8] Y. Leng, Y. Ren, C. Qian and J. Xia, “Code-based preservation mechanism of electronic record in electronic record center of cloud storage,” *Journal on Big Data*, vol. 1, no. 1, pp. 39–45, 2019.

[9] Y. Wang, C. Li, M. Li and Z. Liu, “HBase storage schemas for massive spatial vector data,” *Cluster Computing*, vol. 20, no. 4, pp. 3657–3666, 2017.

[10] Y. Z. Ma and X. F. Meng, “Research on indexing for cloud data management,” *Journal of Software*, vol. 26, no. 1, pp. 145–166, 2015.

[11] L. Tang, Y. Li and J. Qu, “A study of the design of a big-data storage and index model for teaching based on HBase/Spark,” *Journal of Yunnan Minzu University*, vol. 29, no. 5, pp. 486–492, 2020.

[12] Y. Zhang, Y. Z. Ma and X. F. Meng, “Efficient processing of spatial keyword queries on HBase,” *Journal of Chinese Computer Systems*, vol. 33, no. 10, pp. 2141–2146, 2012.

[13] Y. Du, T. Zhang and T. Huang, “A reconstruction method of spatial data using MPS and ISOMAP,” *Journal of Computer Research and Development*, vol. 53, no. 12, pp. 2801–2815, 2016.

[14] H. Meng, M. Z. Zhu and F. Y. Zhang, “Vector spatial database based on hadoop,” *Computer and Modernization*, vol. 22, no. 2, pp. 63–68, 2014.

[15] Y. Li and J. Ma, “The design and implementation of high performance spatial database based on hadoop,” *Journal of Zhongyuan University of Technology*, vol. 25, no. 4, pp. 58–63, 2014.

[16] A. K. Samanta, B. B. Sarkar and N. Chaki, “Query performance analysis of NoSQL and big data,” in *Proc. 2018 4th IEEE Int. Conf. on Research in Computational Intelligence and Communication Networks (ICRCICN 2018)*, Kolkata, India, pp. 237–241, 2018.

[17] X. F. Shi, “Archives Big data storage and retrieval scheme based on distributed NoSQL database,” *Computer Application and Software*, vol. 36, no. 5, pp. 15–20, 2019.

[18] C. Qu, “Design of hybrid cloud storage technology for massive complex heterogeneous data considering heterogeneity,” *Microcontrollers & Embedded Systems*, vol. 21, no. 8, pp. 26–30, 2021.

[19] C. Chen, J. Lin, X. Wu, J. Wu and H. Lian, “Massive Geo-spatial data cloud storage and services based on NoSQL database technique,” *Journal of Geo-Information Science*, vol. 15, no. 2, pp. 166–173, 2013.

[20] Y. Zhang, Y. Wang, Y. Bai, Y. Li, Z. Lv *et al.,* “A new rockburst experiment data compression storage algorithm based on big data technology,” *Intelligent Automation & Soft Computing*, vol. 25, no. 3, pp. 561–572, 2019.

[21] D. Cui and J. Shi, “Implementation of HBase secondary index based on redis,” *Computer Engineering & Software*, vol. 37, no. 11, pp. 64–67, 2016.

[22] M. Gorawski and M. Lorek, “Efficient storage, retrieval and analysis of poker hands: An adaptive data framework,” *International Journal of Applied Mathematics and Computer Science*, vol. 27, no. 4, pp. 713–726, 2017.

[23] Z. Du, “*Research On The Integration Of Unstructured Document Data Storage And Retrieval Technique*,” M.S. thesis, Dept. Computer, Harbin Institute of Technology, Harbin, China, 2015.

[24] N. K. Gundla and Z. Chen, “Creating NoSQL biological databases with ontologies for query relaxation,” *Procedia Computer Science*, vol. 91, no. 1, pp. 460–469, 2016.

[25] Y. Liu, “Research on storage and retrieval of teaching resources based on hadoop,” *Scientific and Technological Innovation*, vol. 21, no. 1, pp. 87–89, 2021.

[26] A. Cuzzocrea, C. K. Leung, B. H. Wodi, S. Sourav and E. Fadda, “An effective and efficient technique for supporting privacy-preserving keyword-based search over encrypted data in clouds,” *Procedia Computer Science*, vol. 177, no. 1, pp. 509–515, 2020.

[27] J. R. Malgheet, E. H. Houssein and H. Zayed, “Efficient privacy preserving of multi-keyword ranked search model over encrypted cloud computing,” in *Proc. 1st Int. Conf. on Computer Applications & Information Security*, Hangzhou, China, pp. 1–6, 2018.

[28] A. R. Li, X. M. Wang, X. L. Wang and B. H. Li, “An improved distributed query for large-scale rdf data,” *Journal on Big Data*, vol. 2, no. 4, pp. 157–166, 2020.

[29] M. N. Torshiz, A. S. Esfaji and H. Amintoosi, “Enhanced schemes for data fragmentation, allocation, and replication in distributed database systems,” *Computer Systems Science and Engineering*, vol. 35, no. 2, pp. 99–112, 2020.

[30] W. Wang, “Research on retrieval based on HBase storage technology of non-relational database,” *Internet of Things Technologies*, vol. 10, no. 1, pp. 103–105, 2020.

[31] H. Jiang, J. Kang, Z. Du, F. Zhang, X. Huang *et al.,* “Vector spatial Big data storage and optimized query based on the multi-level hilbert grid index in HBase,” *Information*, vol. 9, no. 5, pp. 116–137, 2018.

[32] C. Bao and M. Cao, “Query optimization of massive social network data based on HBase,” in *Proc. 2019 IEEE 4th Int. Conf. on Big Data Analysis (ICBDA 2019)*, Suzhou, China, pp. 94–97, 2019.

[33] N. Liu, “Optimazation of genetic algorithms in distributed database query,” *Microcomputer Applications*, vol. 37, no. 8, pp. 108–111, 2021.

[34] S. Vengadeswaran and S. R. Balasundaram, “Core–An optimal data placement strategy in hadoop for data intentitive applications based on cohesion relation,” *Computer Systems Science and Engineering*, vol. 34, no. 1, pp. 47–60, 2019.

[35] D. Koo, H. Yoon and J. Hur, “Secure and efficient data retrieval over encrypted data using attribute-based encryption in cloud storage,” *Computers & Electrical Engineering*, vol. 39, no. 1, pp. 34–46, 2013.

[36] S. C. Wang, I. Pandis, C. Wu, S. He, D. Johnson *et al.,* “High dimensional biological data retrieval optimization with NoSQL technology,” *BMC Genomics*, vol. 15, no. 8, pp. 991–996, 2014.

[37] M. Kiminori, “Functional models of hadoop mapReduce with application to scan,” *International Journal of Parallel Programming*, vol. 45, no. 2, pp. 362–381, 2017.