HBase Write Performance Optimizations

Lu, Haomin

(出版者 / Publisher)
法政大学大学院情報科学研究科

(雑誌名 / Journal or Publication Title)
法政大学大学院紀要. 情報科学研究科編 / 法政大学大学院紀要. 情報科学研究科編

(巻 / Volume)
9

(開始ページ / Start Page)
67

(終了ページ / End Page)
72

(発行年 / Year)
2014-03

(URL)
https://doi.org/10.15002/00010524

HBase Write Performance Optimizations

Lu Haomin
CIS, Hosei University
E-mail:haomin.lu.9z@stu.hosei.ac.jp

Abstract—In the past few years, along with the expansion of the data volume and the cost of computer hardware down, the demand of quickly handling huge amounts of data is driving the rising and development of distributed computing. Database system based on the column storage also gradually rise along with the vigorous development of the cloud computing technology. Many organizations are trying to turn the traditional row storage database migration to the column storage database, so as to adapt to the massive growth of data. In addition to providing a distributed file system and supporting the MapReduce computing framework, Hadoop also provides a scalable and structured data distributed storage system: Hadoop Database(HBase). In currently industry, The HBase column store data is one of the most popular open source products. HBase as an open source implementation of BigTable, the users pay more attention to the performance with the popularization of its application.

Even though the industry has high expectations on HBase, but it still has some disadvantages, for example: not according to the characteristics of the column data stored in columns for efficient data compression, compression mode does not support direct operation data, and so on. Therefore, improving HBase performance has important practical significance. This paper analyses the principle of HBase distributed storage system first, and then studying HBase performance testing in-depth, and outlook on the future development of distributed storage system.

In general, HBase will provide satisfactory read performance as long as the cluster memory is enough. But HBase write performance will be restricted by many factors. Therefore, this paper mainly studies HBase write performance. This paper introduces HBase environment firstly, then analysis of the principle of HBase distributed storage system, focus on HBase writing process analysis. This paper did the random write test and data write test, analyze the test results to find the influence factors for HBase writing performance. By modifying the HBase system configuration parameters, the HBase client and the code of the writing process to achieve the goal of improve HBase write performance. Finally, summary the application of HBase combined with the practical application.

Keywords—Hadoop; HBase; Write performance optimization; Distributed storage system; Database

I. INTRODUCTION

Cloud computing is the latest trend in the development of IT technology, it is received extensive attention by the academia. The development of cloud computing is based on the distributed processing, parallel processing and grid computing technology, cloud computing is a kind of new method of Shared infrastructure. It can self maintain and manage a large of virtual computing resources (including computing servers, storage servers, bandwidth, etc.), in order to provide various IT services. Users can accord to the need to pay when they use the cloud computing services, this is not only reduces the barriers, but also greatly save the cost. Because cloud computing has large potential market. Google, IBM, Microsoft, Amazon, Sun, HP, Yahoo, Oracle and other international well-known companies have been involved in cloud computing. At the same time, cloud computing is beginning to be applied in the telecommunications, finance and other large-scale parallel processing domain[1].

The rapid development of Cloud database market greatly affects the future development direction of database technology, even appeared the dispute that relational database will disappear. At the same time, many issues of cloud database began to be concerned, such as system architecture, data model, transaction consistency, data security and performance optimization and so on. Because HBase is a relatively new research field, there is a few of research to comprehensive introduce the field. Therefore, this paper studied on HBase performance according to the characteristics of HBase distributed storage system, and then put forward the optimization scheme of HBase write performance. HBase is a distributed, scalable, big data storage product based on HDFS, it can be used for online services with huge amounts of data. At the moment, Facebook, Adobe, eBay, Yahoo!,Twitter, TaoBao and other big companies are using it. Company pays more attention to the performance when they choose the system. The performance of HBase is very important as a platform level product, but it can not be explained by a simple testing, it needs based on specific data analysis to close to the practical application[2].

Data is the carrier of information. With the continuous development of information technology, the data becomes more and more important in the modern society. The Data are usually very large in the big Internet companies, so the requirement of database capacity is very high. We adopt traditional relational database like Oracle or SQLServer which can meet the complicated condition query, but we will fall down when deal with terabytes of massive data. Hadoop platform only need to be deployed in ordinary and cheap PC to process PB scale data. So it has very high practical significance and application value because of high efficiency and cost saving.

II. HBASE PROBLEMS IN WRITING

In generally, we can find many places that affect performance through the analysis of data test results. The HBase problems can be found according to the fllowing test. The test objective is to find out the various HBase write performance indicators, analysis the test results and find out

Supervisor: NOBUHIKO KOIKE, Ph.D. Professor the HBase write performance problems to prepare for the write performance optimization. There are two groups of test, one is data writing test, another is random test.

A. Data write test

The HBase cluster include 1 master (namenode) and 5 ResigonServers (datanode), Writing 1 million of data to the cluster (a data about 15K). Insert the same data in HBase and HDFS.

We can see the large difference through the test results, the test results as shown in figure 2.1, data insert time on the HBase is about 10 times of HDFS. Why the performance we insert data into HBase is lower than the HDFS, what is the reason that HBase write performance is so bad. Let's study the insert process. The figure 2.2 shows the write path of HBase.

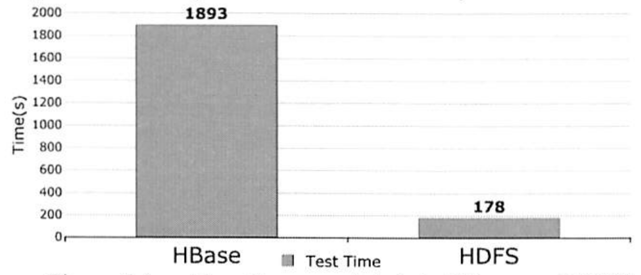


Figure 2.1 writing the same data into HBase and HDFS

The process of data Inserted into HBase is: client requests master before inserting data , master reply which region and which resgionserver can be inserted data, then client and resigionserver communicate to insert data, resigionserver determines which datablock the data inserted into(region is composed by datablock), then stored in HFile in the HDFS (data not necessarily in the local resigionserver). One of the factors that affect the HBase write performance is to use put class buffer. When we use the put class to insert data, the default is writing a data one time by clinet and resigionserver doing a RPC to insert the data. Since there are 1 million data, many times for interprocess communication will affect the time. HBase client provides a write buffer, we do a write operation while the buffer is filled, thus reducing the number of writing.

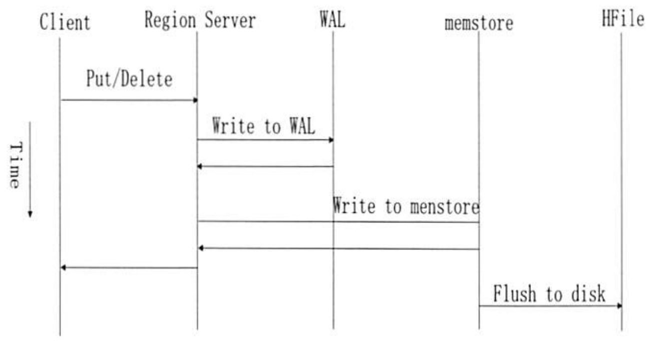
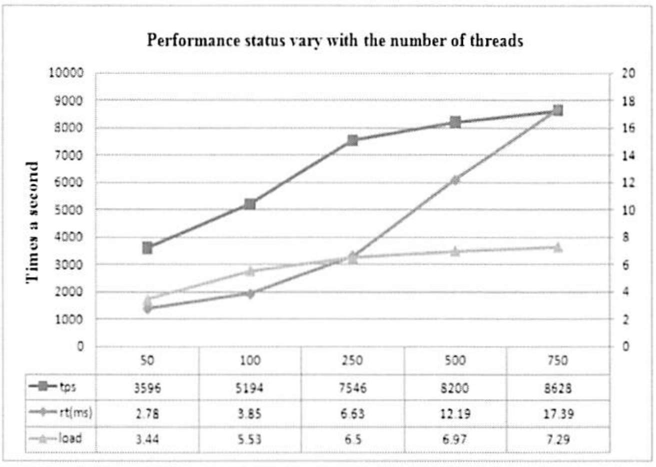


Figure 2.2 HBase write path

B. Random write test

we can see from the random write test results as shown in figure 2.3, when the client threads at around 250, the response time is about 6ms at this point, tps is around 7.5k, it is almost the best state. As time goes on, the performance begins to



	50	100	250	500	750
tps	3596	5194	7546	8200	8628
rt(ms)	2.78	3.85	6.63	12.19	17.39
load	3.44	5.53	6.5	6.97	7.29

Figure 2.3 random write performance test results

In the random write test, we can see some problems:

a) TPS falling is quite obvious with single machine region number changed more.

b) When "hbase.Regionserver.Handler.Count" is 100 (the default is 10), 100 thread will be blocked when pressure is big enough, when the number of thread increase to 300 almost enough, the TPS also reach bottleneck;

c) When datanode is few, lead to write tps low, maybe the reason is the compact will consume too much network IO;

d) When we use gz compression that will cause heap memory leak;

e) When the pressure and region increase, split and flush will cause larger influence to stability of writing. When memory is enough, we can adjust the split file size and memstore flush size, according to the situation to decide whether it can be adjusted or not.

C. Summary

Through the analysis of the test results, we can see the main factors influence for writing are requesting number uniform distribution, Blocking Update and Delaying appear or not, the number of flush , HLog, DataNode, Split File Size and so on.

III. Current HBase and Problems

We found a lot of places which influence the HBase performance through the above analysis. why they are so? For the academic community, HBase wants to provide rich functionality like DBMS, such as query, index and transaction processing, there are still many issues required to be solved. But we can find the reasons from the current HBase system.

This chapter analyses the architecture and read and write process of HBase. Our understandings of the performance problems will be more clearly after understanding the current HBase.

A. HBase architecture

HBase is an open source implementation of BigTable, the architecture is similar to BigTable. Figure 3.1 shows the architecture, the HBase architecture including Client, Zookeeper, Master, RegionServer and Store, specific functions are as follows:

1)Client: Contains access interface of hbase, client maintains some cache to speed up access to hbase, such as the regione location information.

2)Zookeeper: a)Ensure that there is only one master in the cluster at any time. b) Store all addressing entrance of Regions. c)Monitor Region Server state real-time, send notification of region server online and offline information to Master real-time. d) Store schema of Hbase, including table, column family of each table.

3)Master: a) Distribute region for region server. b) Responsible for the region server load balancing. c) Find failure region on the server and redistribute it again. d) Recovery junk fileson GFS. e) Processing schema update request

4)Region Server: a) Region server maintains regions that Master assigned to it, dealing with the I/O requests of egion. b) Region server responsible for Region segmentation in the running process which becomes too large.

5)Store: Store is the core of HBase storage, is composed of MemStore and StoreFiles. The user written data will put into MemStore, data will be stored in a StoreFile when the MemStore is full, the StoreFile will be stored in the HFile of HDFS files system[3].

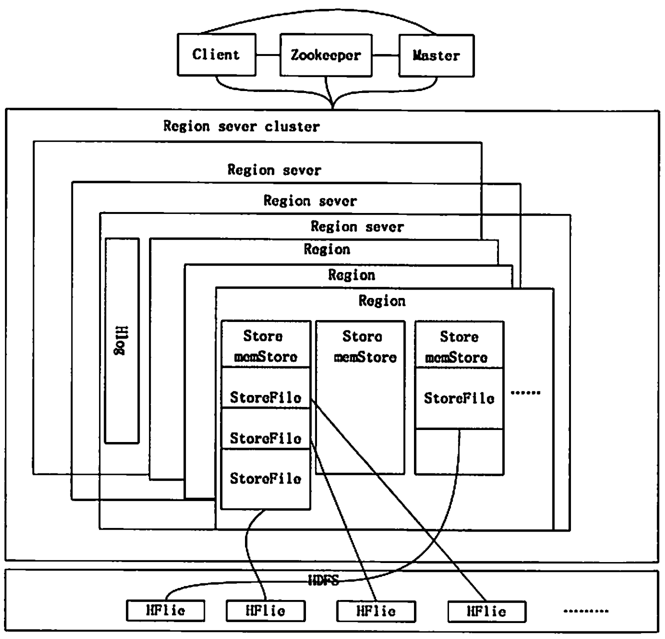


Figure 3.1 HBase architecture

B. HBase read and write process

As mentioned above, HBase uses the MemStore and StoreFile to update tables. Data is written to the Log(WAL log) and memory (MemStore) firstly , the data in MemStore is sorted, when MemStore accumulate to a certain threshold, it will create a new MemStore, and the old MemStore is added to the flush queue, flush to disk to become a StoreFile from a separate thread. At the same time, the system will record a redopoint in zookeeper, said the moment before changes are persistent. When the system accident occurred, may lead to the data in memory (MemStore) loss, using the Log (WAL log) to restore checkpoint data in this time. The previously mentioned StoreFile is read-only, once created can not be modified. Therefore the HBase update is actually increasing operation. When a Store StoreFile reached a certain threshold, it will carry out a combined (major compact), will be on the same key revision merged together, forming a large StoreFile, when reaching a certain threshold size of StoreFile, and split for the StoreFile, divided into two StoreFiles. Because the table is updated continuously add, handle read requests, the need for StoreFile and MemStore all access to Store, they will be in accordance with row key to merge, because StoreFile and MemStore are sorted, and StoreFile with memory index, the process of the merger is still relatively fast[4].

C. Summary

This chapter analysed HBase in-depth. We understand the principle of the causing low write performance reasons by learning current HBase. This made a theoretical preparation for the following optimizations.

IV. HBASE WRITE OPTIMIZATIONS AND EXPERIMENTS

We found some bottlenecks of the system through the analysis of random test and data test above. This chapter is mainly about optimization. The following is system level parameter configuration optimization and client design optimization to solve these problems. Test optimization effects are shown through the HBase cluster.

A. Test environment

1) Software environment

ubuntu 12.04 LTS; hadoop-1.1.2; HBase0.94.8

2) Hardware environment

HBase cluster have 6 machines, one as Master, the other 5 as the region servers, physical memory is 24G, they are in the same room and connect each other through router.

3) Test data description

Data Type: One month access log of a Chinese video site.

Data Structure: A row of nine columns, including user access page, keywords and other user informations.

Data Size: Data is 1 million(each data about 15K), the total is about 15G.

4) Test method

Each test is divided into 3 groups, the first group startups 1 regionserver, the second group startups 3 regionservers, the third group startups 5 regionervers. Each test is consists of configuration modified before and after. Each test is written into 1 millions of data to the HBase cluster (a data is about 15K), the abscissa represents different case, the vertical represents the write time.

B. Cancel automaticflush

1) Configuration

Cancel the automatic flush, the configuration content is shown below:

```
htable.setAutoFlush(false);
```

2) Test results

Region Server	1	3	5	Average Time(s)
Default	3211	2601	1901	2571
Modified	1812	1117	903	1277.3

3) Principle

If we set autoflush=false, When client submits delete or put request. Firstly, the request will be cached in the client until the data over 2M (the default size of hbase client write buffer), after that requests are submitted to regionserver. regionserver returns information to client after receiving delete or put request. In such a situation, the communication between client and regionserver is intermittent. As shown in figure 4.1. The default is automatic flush, the request communication between client and regionserver will not stop, every request will be sent to regionserver. The first thing what the regionserver to do is to write Hlog when the regionserver receives the request. This spends a lot of time.

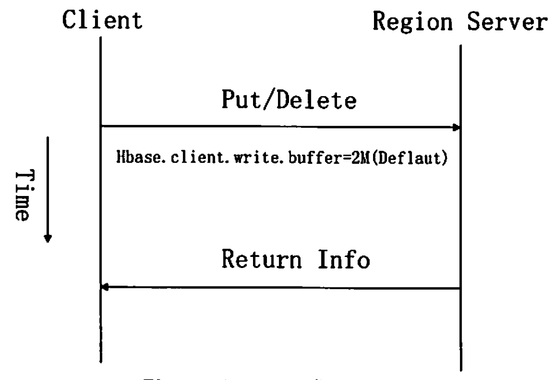


Figure 4.1 auto flush path

4) Observation

In order to improve the writing speed of HBase, many developers set autoflush=false. But we need to be careful in the situation. The reason is: when we set the autoflush=false, if the cache can't get to the size we set and client crashes in this time. Data can not be sent to regionserver and it will be lost. Tt is can't be accepted for the zero tolerance online services.

C. Optimize the write buffer size

1) Configuration

In the situation of canceling auto flush, adjust the buffer size to improve write performance. The buffer size in the code is set 6M. The configuration content is shown below:

```
HTable htable = new HTable(config, tablename);
htable.setWriteBufferSize(6 * 1024 * 1024);
htable.setAutoFlush(false);
```

2) Test results

Region Server	1	3	5	Average Time(s)
Default(2M)	1012	1117	903	1277.3
Size=6M	782	690	597	689.6
Size=20M	2419	3127	2670	2738.6
Size=100M	3270	4012	3793	3751.6

3) Principle

HBase Client send request to regionserver until the buffer accumulating to the set size. It can reduce connection times between client and regionserver. But the buffer size is not the bigger the better. On the one hand, we set auto flush false before setting the buffer, the data will be lost when client cashes (mentioned above); on the other hand, although the connection times reduced, but if buffer size is too large, that will spend a lot of time when the data accumulated to the set size. Therefore, we need to set different buffer size in different applications. In This data test, we found 6M is the best.

4) Observation

We set the buffer size after setting auto flush false, then setting the write buffer size for 100M, 20M, 6M. We can see the write time obviously improved when the buffer size is 6M. HBase client sends request to regionserver until the data accumulated to set buffer size, the benefit is reducing the number of connection times. But why the write time is longer when we set 20M and 200M, the reason is the data are accumulated to the set size need a lot of time. Therefore buffer size is not the bigger the better, which size will be set need consider different situations.

D. Optimize the number of RPC handler

1) Configuration

Increasing the number of RPC (Remote Procedure Call) handler by modify the HBase regionserver handler count in hbase-site.xml. The default count is 10. We changed it to 100 and 500. The configuration content is shown below:

```
<property>
<name>hbase.regionserver.handler.count</name>
<value>100</value>
</property>
```

2) Test results

Region Server	1	3	5	Average Time(s)
Default(10)	3211	2601	1901	2571
Count (100)	2313	1678	1501	3913.3
Count(500)	3609	4012	4119	1830.6

3) Principle

The configuration defines the number of each regionserver RPC Handler. Regionserver receives and processes external request through RPC handler. So if we increase the number of RPC handler we can improve the ability of HBase receiving and processing the request.

4) Observation

The write performance improved when the handler number is 100, but when the handler number is set to 500, the performance decrease. Because the handler number is not the bigger the better, that depends on the hardware of node. Therefore, setting the right number of handler needs to consider the hardware of different situations.

E. Cancel the WAL(write ahead log)

1) Configuration

Cancel the WAL(write ahead log), The configuration content is shown below:

```
Put put = new Put(rowKey);
put.setWriteToWAL(false);
```

2) Test results

Region Server	1	3	5	Average Time(s)
Default	3211	2601	1901	2571
WAL(false)	2801	2019	1601	2140.3

3) Principle

HBase will record operation information in WAL before writing data, then use the memstore to temporarily store data, sort the data, in the last write to the HFile. As shown in figure 4.2. WAL records the operation information of memory, we can reduce the disk seek time if we close WAL, so the time become short.

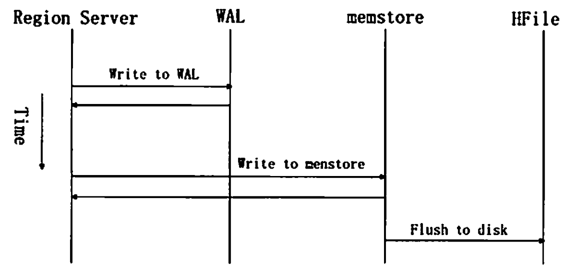


Figure 4.2 data flush path

4) Observation

We can improve the write performance by shutting down the WAL(write ahead log). But in generally, we do not recommend improving the writing performance by closing WAL, because HBase will put operation information to WAL before writing data to ensure HBase can based on the WAL record to recovery data in abnormal conditions.

F. Optimize the number of compaction thread

1) Configuration

Increase the value of attributes below to raise the number of compaction thread. The default value is 1, set it to 2. The configuration content is shown below:

```
</property>
<property>
  <name> hbase.regionserver.thread.compaction.small
</name>
  <value>2</value>
</property>
<property>
  <name>
hbase.regionserver.thread.compaction.large</name>
  <value>2</value>
</property>
```

2) Test results

Region server	1	3	5	Average Time(s)
Default(1)	3211	2601	1901	2571
Modified(2)	1568	1391	891	1283.3

3) Principle

If we open the HMaster web to check the number of request for each region server when we are inserting data . We'll find out the request number some regionservers receive is 0 sometimes. Maybe the regionserver is compacting the data lead to this situation. Compaction process will block the writing speed. Therefore, increase the number of threads of compaction can improve the processing speed and speed up the process of writing[5].

4) Observation

When we find the request number which some regionservers received is 0 sometimes. Maybe the reason is that client doesn't send request to regionservers. Therefore, we should ensure the write request in the regionservers is roughly average distributed in most of the time when we check the number of request in each regionserver through HMaster web. In such a situation, we consider the compaction.

G. Create empty region advance

1) Configuration

By default, the size of region is 256MB. We can adjust the value through xml code "hbase.hregion.max.filesize" in the hbase-site.xml. The test data is 1 million(each data about 15K), that needs 58 regions to store. The number of region is about 1/3 to 1/2 of the total amount according to the experience. So we created 30 empty regions in advance. The following is an example of creating region in advance.

```
public static boolean createTable(HBaseAdmin admin,
HTableDescriptor table, byte[][] splits)
throws IOException {
try {
admin.createTable( table, splits );
return true;
} catch (TableExist***ception e) {
logger.info("table " + table.getNameAsString() + " already
exists");
// the table already exists...
return false;
}
}
```

2) Test results

Region server	1	3	5	Average Time(s)
Default	3211	2601	1901	2571
Create region advance	1568	1391	891	1283.3

3) principle

HBase takes some measures to sort when HBase write data, this is the merging and splitting mechanism of HBase. The basic unit of HBase expansion and load balancing is region. Region is a collection of rows. When region reach a certain size, the region will split automatically. For a table (HTable), there is only one region fist. When the data scale increase, the system will monitor this table to ensure that data will not over configured size. If system find that the table size over the limit, the region will be divided into two regions which size are roughly equal. Therefore, creating regions according to the amount of data and row key rules in advance can greatly reduce the number of region split times[6].

4) Observation

We can see the writing performance improved a lot through the test data. In order to improve writing performance, HBase official advises to create region in advance, this concept is called pool. But the number of region is not the more the better, because too much region can also reduce performance. The number of region is about 1/3 to 1/2 of the total amount according to the experience. In most cases, the region split has great influence to write performance. We can reduce region split from creating region in advance, there is another way, that is to increase "hbase.hregion.max.filesize". Increasing the size of region file reasonably can also reduce split times, improving writing performance.

H. Summary

We can improve the performance through system configuration parameters adjustment. In addition, the design of client also has great influence to the writing performance. We reached the purpose of optimization by adjusting system parameters and client design. In addition, there are many other places can be optimized to improve the write performance. For example, uniform distribution write pressure in each regionserver and adopt distributed manner to insert data into the program and so on, are also belongs to optimization content.

V. CONCLUSION

This thesis introduced HBase framework, characteristics, model, data access method and key algorithm, detailed analyzed of the HBase write path. Based on the understanding of the HBase, then do the HBase performance optimization. Because of the performance of HBase is mainly limited to the writing process, so this paper did a large number of optimization about writing performance according to these characteristics. We got satisfactory results in the last.

This paper found some problems in the process of HBase data write test and random testing, we found the reason through the analysis of performance problems. We discovered problems and make optimization on the system configuration and design of client. The optimizations mainly include the cancel automatic write, close WAL, adjust buffer size, increase the count of RPC handler and compaction thread, and create the region in advance. After the optimization, we did the test to prove the optimization effective. Provide a basis for the research and application of HBase.

In this paper, the optimization of HBase is not completely, although read performance of HBase will have high efficiency when there is enough memory, but there are still some spaces to improve. In addition, there are still issues affect performance in many places in the system configuration, the client design and the logic structure. And the optimization is different in different situations, it is still need to be pay attention to. We hope we can continue to study these problems in the future.

VI. ACKNOWLEDGEMENT

In the process of writing thesis, I got a lot of help from other people, so that I can complete thesis successfully. First of all, I would like to thank my supervisor professor Nobuhiko KOIKE. This paper is completed under his guidance. In addition, I got many advices from my classmates and other professors, I also sincere thank all of them.

VII. REFERENCES

[1] Haijie Ding,Yuehui Jin,Yidong Cui,Tan Yang. DISTRIBUTED STORAGE OF NETWORK MEASUREMENT DATA ON HBASE[A]. 2012:5.

[2] 唐常杰,熊民. The Temporal Mechanisms in HBase[J]. Journal of Computer Science and Technology,1996,04:365-371.

[3] Shengmei Luo,Qing He,Lixia Liu,Xiang Ao,Ning Li,Fuzhen Zhuang. Parallel Web Mining System Based on Cloud Platform[J]. ZTE Communications,2012,04:45-53.

[4] HBase, http://hbase.apache.org/

[5] Jun-Ki Min,Mi-Young Lee. DICE:An Effective Query Result Cache for Distributed Storage Systems[J]. Journal of Computer Science & Technology,2010,05:933-944.

[6] Shi Huijun,Rao Ruonan. Scalable Distributed RDFS Reasoning using MapReduce and Bigtable[A]. IEEE.Proceedings of 2011 4th IEEE International Conference on Computer Science and Information Technology(ICCSIT 2011) VOL03[C].IEEE:,2011:4.