Hbase와 HDFS에서 데이터 검색 시간 비교

신 응억 뉘엔,퀴엣 뉘엔 반, 김경백
전자컴퓨터공학부
전남대학교

Comparison the query time of searching data on HBase and HDFS.

Sinh Ngoc Nguyen, Van-Quyet Nguyen, Kyungbaek Kim
Dept of Electronics and Computer Engineering,
Chonnam National University
e-mail : sinhngoc.nguyen@gmail.com, quyetict@utehy.edu.vn, kyungbaekkim@jnu.ac.kr

요 약

Nowadays, handling data efficiently plays a significant role in big data related companies. Because of data explosion, we need to have advanced efficient solutions in searching data, processing data, and obtaining statistics of data. Currently, Hadoop and Spark are the leading frameworks in big data processing. In the aspects of storage, there are several frameworks providing big data distributed storage such as HDFS and HBase. But, each of them gives the different performance in handling particular data under different situation. In this paper, we compare the performance of searching query in agricultural big data which stored on HDFS and HBase in aspects of sequential searching and random searching. To compare the data searching performance, we compute the summation of area of selective farms, which obtained by a searching query. Through the implementation based evaluations, we observed that the time for querying data stored in HDFS is shorter than HBase in the case of sequential searching. In contrast, we found that random searching with HBase gives better performance than HDFS. We found the main reason of this performance difference as the mechanisms provided by HBase such as the column-family data handling mechanism and low latency access row mechanism. These mechanisms focus on the row or column which we want to access in random, and it reduces the volume of data in memory for reading, then Hbase achieves shorter query time in random searching.

1. Introduction

The data explosion has overwhelmed several systems of companies with huge of data volumes and analytics. With the daily growing of data, it creates a fast increase of demand in storage, statistic and processing. It has an importance how to access quickly of information in big data. To archive this purposes, we need to have an advance solution for searching and querying in big data.

At the current, Hadoop [1] and Spark [2] are the leading frameworks in big data processing. Hadoop provides MapReduce model to process huge amounts of data in parallel. After each phase of processing, it spends more time to read and write data to hard disk, so that is the limitation of Hadoop. In the other, Spark is a novel framework to resolve above problem. That is an open source cluster computing framework developed at AMPLab. It provides an application programming interface called RDD with in-memory processing. So that Spark is better than Hadoop in iterative data processing.

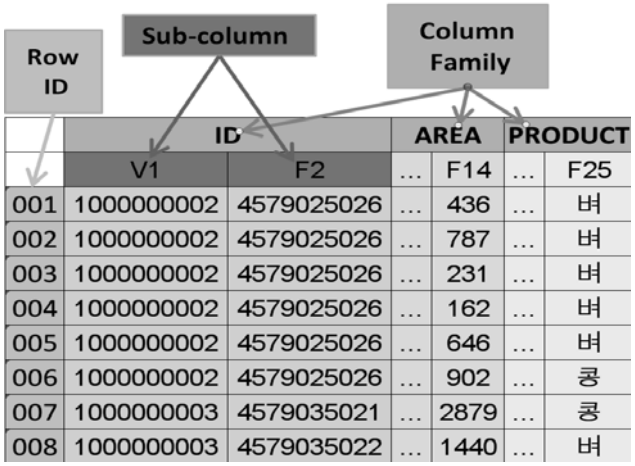
Also, data storage plays an important role in big data system. That is the restricted of Relational Database Management System to store Bid Data. Currently, Hadoop provides Hadoop Distributed Files System (HDFS) [4] for big data storage. It enables scalable and reliable data storage, and it was designed to span large clusters of commodity servers. The other of storage framework is Hbase [5]. That is a data model designed to provide quick random access to huge amounts of structured data. It is built on the top Apache Hadoop and become an increasingly popular database choice for applications that need fast random access to large amounts of data

In this paper, we compare the performance of searching query on dataset, which stored on HBase and HDFS. And using Spark RDD [6] to compute the total land of a farm after having the result of above searching. We made two experiments to show the performance of random searching and sequential searching. The first is calculating the total acreage of a farm given by Farm_ID, and the second is calculating total acreage of all farm in dataset.

2. Background

A. Hadoop Distributed Files System

Hadoop Distributed Files System is a distributed file system to store large amount of data. It designed to overcome large cluster of commodity servers, provide the scale, and reliable data storage. It is fault-tolerant, distributed storage system that works closely with a wide variety of concurrent data access applications. In Hadoop system, HDFS coordinated by YARN [7] to enables the combined storage use for computation across many servers.



	ID			AREA		PRODUCT
	V1	F2	...	F14	...	F25
001	1000000002	4579025026	...	436	...	벼
002	1000000002	4579025026	...	787	...	벼
003	1000000002	4579025026	...	231	...	벼
004	1000000002	4579025026	...	162	...	벼
005	1000000002	4579025026	...	646	...	벼
006	1000000002	4579025026	...	902	...	콩
007	1000000003	4579035021	...	2879	...	콩
008	1000000003	4579035022	...	1440	...	벼

Figure 1 Data organized in HBase

B. HBase

As we know, Hadoop provide distributed file system for storing large volume of data. It is access only data in a sequential manner. It mean that searching data must run on entire data even for simple of job. So it is not good in case of random access. HBase is a new solution to access any point of data in a single unit of time. HBase is a data model provides quick random access to huge amount of structured data in distributed system. It is also an open source project built on top of Hadoop.

HBase provides column-oriented and row-oriented mechanism. So it manages data through Column-Family and Row_ID. Each Column-Family includes sub-column as the column in SQL database. Row_ID is an identity of each row. It looks like the index in SQL, used for random data access.

C. Spark

Spark is a leading framework in big data processing. It is an open source and build around speed, easy to use, and sophisticated analytics which provided by AMPLab. Hadoop is restricted by execution time, it need to read and write data from memory into disk after each phase. So Spark provides resilient distributed dataset (RDD) with in-memory processing to reduce the time of saving data. RDD is a data structure which provide data in distributed over cluster of machine. RDD facilities the implementation of iterative algorithm that visits their dataset multiple times in a loop, and interactive data analysis.

3. Performance Evaluation

In this work, we made the experiment to compare the query time of searching data which stored on HDFS and the other stored on HBase. In order to search data which stored on HDFS, we need to read sequentially and get line by line for searching. With data store on HBase, that is a random searching case. We focus on the row to get data provide by RowID.

A. Datasets.

The dataset includes information as show in agriculture which collected by Korea Government in 2015 named Nongjak.csv, which shows in Figure 2. The dataset describe information in farm such as the acreage of fields in farm, how many kind of agricultural product in farm, information about farmer, and so on. The size of dataset around 2GB. It has more than 11 million of records, with 34 fields.

V1	F2	...	F14	...	F25
1000000002	4579025026	...	436	...	벼
1000000002	4579025026	...	787	...	벼
1000000002	4579025026	...	231	...	벼
1000000002	4579025026	...	162	...	벼
1000000002	4579025026	...	646	...	벼
1000000002	4579025026	...	902	...	콩
1000000003	4579035021	...	2879	...	콩
1000000003	4579035022	...	1440	...	벼

Figure 2 : Nongjak Dataset

We just explain several fields related to our experiment as below:

V1 - that uses to identify farms.

F2 - the addresses code of farms.
F14 - This field describes the acreage of each field in farm.
F25 - in a farm, there are several kinds of agricultural product such as rice, bean, potato, and so on. This field indicate the kind of plan in the farm.

B. Implement searching query

Method 1: Read raw data on HDFS and calculate total acreage of land.

Require: *ID_Farm* is the ID of farm we want to search and calculate the total acreage. *V1* is column includes *ID_Farm* in Nongjak.csv. *F14* is column includes the acreage of each field in farm.

```
1:  /*Read data from Nongjak.csv*/
2:  dataset = ctx.textFile(Nongjak.csv);
3:  /*Filter to get record given at Farm_ID*/
4:  filter_data = dataset.filter(){
5:        return (dataset.split(",")[V1].contains(ID_Farm)
6:  };
7:  /*Use Maper to split data to (key,value)*/
8:  maper_data = filter_data.mapToPair{
9:        return new Tuple<key, value>(V1, F14);
10: };
11: /*Use Reducer to group value with the same key*/
12: result = maper_data.reduceByKey(){
13:       return new Tuple2<Float>(F14i+F14j);
14: };
15: /*Save result to HDFS*/
16: result.saveAsHadoopFile();
```

In this paper, we compare the query time of searching and calculating the total acreage of a farm from dataset which stored on HBase and HDFS.

First case, dataset is stored as raw data on HDFS. We read file Nongjak.csv on HDFS into memory using Map/Reduce, then process line by line to get the key/value pair that are ID and acreage of a farm. In the Reducer task, each key we can calculate the total acreage of a farm. We can illustrate the flow of calculation at Method 1.

In the second case, dataset stored in HBase model. HBase provides reading mechanism with column-oriented and row-oriented mechanism. Instead of reading all dataset, it focuses on the column or row which wants to read. HBase organize data in columns family. Each family column includes columns in dataset. So it does not need to read all data into memory that read the row or column which is needed for processing. After having the result at above reading step, we use map and reduce in Spark to calculate the acreage of farm which summary as Method 2.

Method 2: Read data stored on HBase table and calculate total acreage of land

Require: *ID_Farm* is the ID of farm we want to search and calculate the total acreage. ***ID*** is column family in HBase includes ***V1***, which is contain ***ID_Farm***. ***AREA*** is column family in HBase include ***F14*** which contains the acreage of each field in farm.

```
1:  /*Get Row_ID from ID_Farm*/
2:  RowIDs = getRowID(ID_Farm);
3:  /*Initiate a HBase table*/
4:  HBtable = new HTable(config, "nongjak");
5:  /*Read rows depend on RowID*/
6:  List<String> HBdata = new ArrayList<String>();
7:  foreach (row in RowIDs[]) {
8:        id_farm = getValue(Bytes.toBytes("ID"), Bytes.toBytes("V1"));
9:        acreage = getValue(Bytes.toBytes("AREA"), Bytes.toBytes("F14"));
10:       HBdata.add (id_farm, acreage);
11: }
12: maper_data = HBdata.mapToPair {
13:       return new Tuple<key, value>(V1, F14);
14: }
15: result = maper_data.reduceByKey() {
16:       return new Tuple2<Float>(F14i+F14j);
17: }
18: result.saveAsHadoopFile()
```

C. Result

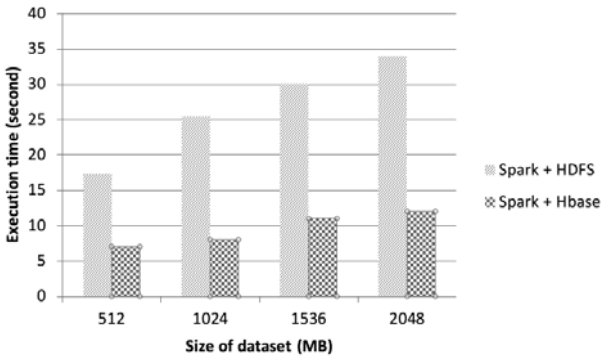


Figure 3 : Random Case Performance Evaluation

In the Method 1, we read data from Nongjak.csv file stored on HDFS and process to calculate the total of acreage given by a ID_Farm. In order to do that, we must read all row of dataset and check the matching of ID_Farm to get the needed data. So it spends much time for reading step. In another case, we use HBase API to read data stored on HBase. It provides column-oriented and row-oriented mechanism for reading. So it just focuses on the row given by RowID. It does not to read all data in table into memory as Method 1 had do. Figure 3 shows the graph of duration time for reading and calculate the acreage given by ID_Farm. Using Spark and HBase is faster than Spark and HDFSin case of random reading data.

In the other experiment, we calculate the total land of each farm in dataset. In this case, we need to read all dataset for calculation. So storing data in HDFS have a better result than HBase in case of sequential searching. Figure 4 shows the result of sequential case

of searching.

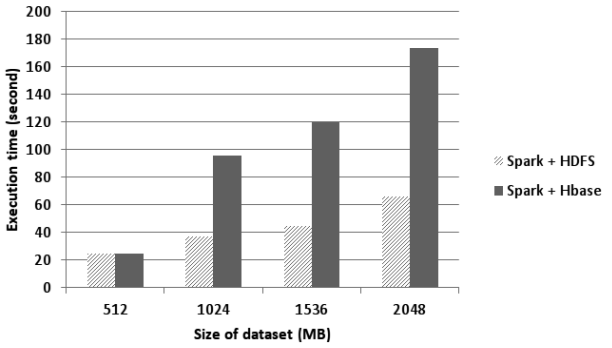


Figure 4: Random Case Performance Evaluation

3. Conclusion

In this paper, we evaluate the performance of accessing data which stored in HDFS and HBase in case of sequential accessing and random accessing. Data stored in HDFS provides the sequential accessing better than HBase. In contrast, random accessing on HBase is better than HDFS. So it depend on the purpose of user to design database and select the appropriate storage system to get the best performance.

Acknowledgements

This work was supported by the National Research Foundation of Korea Grant funded by the korean Government(NRF-2014R1A1A1007734). This research was supported by the MSIP(Ministry of Science, ICT and Future Planning), Korea, under the ITRC(Information Technology Research Center) support program (IITP-2016-R2718-16-0011) supervised by the IITP(Institute for Information & communications Technology Promotion).

References

[1] Hadoop, Apache. "Apache Hadoop." URL http://hadoop. apache. org (2011).

[2] Spark, Apache. "Apache Spark Home Page, retrieved from internet on 23 September 2014." (2014)

[3] Karun, A. Kala, and K. Chitharanjan. "A review on hadoop—HDFS infrastructure extensions." Information & Communication Technologies (ICT), 2013 IEEE Conference on. IEEE, 2013.

[4] Vora, Mehul Nalin. "Hadoop-HBase for large-scale data." Computer science and network technology (ICCSNT), 2011 international conference on. Vol. 1. IEEE, 2011.

[5] Zaharia, Matei, et al. "Spark: cluster computing with working sets." HotCloud 10 (2010): 10-10.

[6] Vavilapalli, Vinod Kumar, et al. "Apache hadoop yarn: Yet another resource negotiator." Proceedings of the 4th annual Symposium on Cloud Computing. ACM, 2013