Similarity Search on Massive Data Based on FPGA

Yanzheng Wang, Hong Gao, Shengfei Shi, and Hongzhi Wang(✉)

School of Computer Science and Technology,
Harbin Institute of Technology, Harbin, China
{yz_wang, honggao, shengfei, wangzh}@hit.edu.cn

Abstract. Data quality is a very important question in massive data process. When we want to distill valuable knowledge from a mass set of data, the key point is to know whether the dataset is clean. So before we extract useful massage from the dataset we'd better do some data clean job. Similarity search is a very important method in data clean. MapReduce will be used to do similarity search in our data clean system. But the efficiency is very low. We found that when we process the massive data stored in HDFS with MapReduce programing model every part of the dataset will be scanned and this is very time-consuming especially for large scale dataset. In this paper we will do filter operation on original data with hardware before we use similarity search to do data clean.

Keywords: Data clean · FPGA · Similarity search · MapReduce

1 Introduction

There is growing enthusiasm for the notion of "Big Data". More and more people want to find treasure from "Big Data". However data quality issues will result in lethal effects of big data applications. Therefore clean the massive data with the problem of quality is very important. Real treasure will be found only the data quality issue is taken seriously [1].

In traditional relation database, multi-tuples representing the same entity is the most common type of poor-quality data. Organizing the multi-tuples which represents the same entity is an effective method of management of poor-quality data. A Similarity search [2–4] problem is that given a query, one can get a list of results and each pair of them meets the similarity threshold. Similarity search is a very important technique in massive data clean.

In order to clean large amounts of data, we use MapReduce [5] to do similarity search on massive data stored in HDFS [6]. MapReduce is a part of Hadoop environment and it is a programming model for processing large data sets with a parallel, distributed algorithm on a cluster.

Although the performance is very good when we do similarity search with MapReduce on massive data, it becomes slow as data size stored in HDFS grows fast. MapReduce will scan every part of the table stored in HDFS and this is very

H. Gao et al. (Eds.): DASFAA 2016 Workshops, LNCS 9645, pp. 343–352, 2016.
DOI: 10.1007/978-3-319-32055-7_28

time-consuming. In order to fix this problem we will use FPGA [7] to filter the original data. And FPGA does better in this job.

The main contributions of the paper are summarized as follows:

1. Put FPGA into hadoop environment and call FPGA to do filtration job. In order to use FPGA in hadoop, we changed Mapreduce programing model.
2. Two algorithm was proposed and implemented. A lot of experiments was performed to test our system.

The rest of the paper is organized as follows: Sect. 2 describes the background of our work. We present two algorithms in Sect. 3 and we did some explain for each of them. In Sect. 4, we give the results of our experiment evaluation. Finally, Sect. 5 concludes this paper.

2 Background

2.1 Similarity Search

Due to data reported many times or other human factor, it's quite normal for data repeat in real work environment. Field similarity was used to judge repeated data. The similarity factor S ($0 < S < 1$) between two fields represent the level of similarity. It is calculated according to the content of the fields. The smaller of S, the similarity between the fields. $S = 0$ means the two fields completely same. The method to calculate S is different according to the field type.

For bool type, if the two fields are equal, S is zero; otherwise, S is one.

For numeric field, we use relative difference to get the similarity factor. It can be represent by

$$S(S\_1, S\_2) = |S\_1 - S\_2| / \max(S\_1, S\_2) \tag{1}$$

For character type, there is a relatively easy method to calculate the similarity factor. Divide the number of matching character by the average number of the two character string.

$$S(S\_1, S\_2) = |K| / ((|S\_1| + |S\_2|)/2) \tag{2}$$

In this formula, K is matching character of the two character string.

Set the threshold and discover similar objects with similarity search. Then we can do delete operation or other data clean operations.

2.2 MapReduce Programing Model

In recent years Hadoop was used to solve massive data problem due to its distributed file system and MapReduce programing model. MapReduce is a software framework capable of processing large amounts of data-sets in parallel across a distributed cluster of processors or stand-alone computers. A MapReduce program is composed of two procedures:

- **Map**() procedure performs filtering and sorting
- **Reduce**() procedure performs a summary operation

2.3 System Architecture

Our data clean system is based on Hadoop environment, data-sets stored in HDFS (Hadoop Distributed File System) and processed with MapReduce. In order to speed up similarity search module, we use FPGA to do filter operation instead of CPU. We can see FPGA does better than CPU in this job in many previous papers. As we write before, Map procedure performs filtering and sorting. So we will add FPGA into Map procedure to do filter job (Fig. 1).

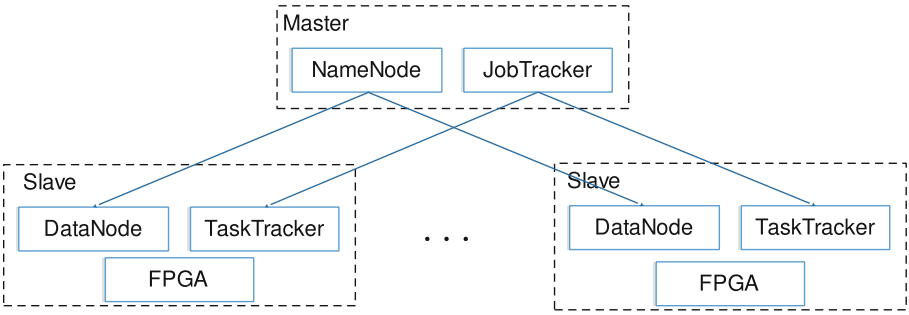

Fig. 1. Hadoop system with FPGA. As we can see from this architecture, FPGA will insert into each slave.

When we use MapReduce to do data clean job, we will change that job into another form which can be done with FPGA.

2.4 File Format

ORCFile [8, 9] was introduced in Hive 0.11 and each ORCFile is composed of one or several stripes. The default size of stripe is 250 MB. Stripes have three sections: a set of indexes for the rows within the stripe, the data itself, and a stripe footer.

This file format will be used as default file format in our data clean system because of its excellent compression. This file format is convenient for hardware to process (Fig. 2).

The stripe footer contains the encoding of each column and the directory of the streams including their location. In row data each column is stored separately. Index data includes min and max values for each column and the row positions within each column. We will use the statistic information of each column stored in index data to do coarse filtration.

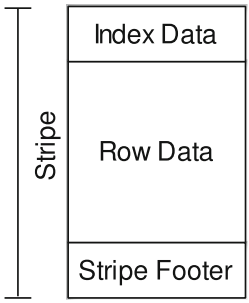

Fig. 2. Stripe's structure, ORCFile is composed of stripes.

3 Filter Operation

ORCFile was used as our default file format. Our dataset was stored in HDFS in this format. Each file stored in HDFS will be sliced into several ORCFiles and every ORCFile composed by some stripes. Stripe will be processed as a whole, it won't be split at here. This operation was added into map function. Map function was changed by us to process with FPGA (Fig. 3).

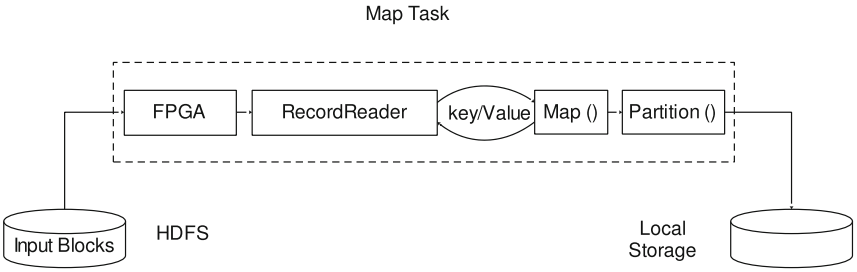

Fig. 3. Map task's execution with FPGA

3.1 Coarse Filtration

If we just use a part of data of one file and the rest of the file has nothing to do with the result, we do not want to scan all of the file. If we can read the data related the result only we can avoid many I/O time-consuming.

Stripe has some index data, we use this information to do coarse filtration. When we store data file in HDFS, program will calculate statistical information for each column. Each stripe has statistical information for its row data. This information will be used for coarse filtration.

Algorithm 1. Coarse Filtration

Input: *sinfo* – Stripe Information
Input: *ficon* – filter conditions
Output: *true* iff this stripe can be ignored without read

```
1: Function coarseFil(sinfo, ficon)
2: if we want to use FPGA to do filter job then
3:     for each filter condition c ∈ ficon do
4:         if the variable in condition c can be used to do filtration then
5:             get s ∈ sinfo
6:             result ⇐ compare(c, s);
7: update the ficon with result;
8: for each operator op ∈ ficon do
9:     if op is 'a|b' then
10:        return a&b;
11:    if op is 'a&b' then
12:        return a|b;
```

For every stripe, we can get its statistic information by its id. Then the information will be used by function coarseFil. In function coarseFil we can decide whether this stripe needs to be read from disk through compare stripe's statistic information with filter conditions. If one stripe needs not to be read from disk, we can avoid I/O operations on the stripe. This can save our time.

We analyze the filter condition and compare the data user defined with every stripes statistic information in Line 2–6. After we will replace the filter condition with the compare result. And the final result will be calculated in Line 8–12.

3.2 Add Hardware into Software

In this paper, we use FPGA to do filtration job. We expect FPGA will give us a good performance in this kind of job. In order to use FPGA in hadoop environment we will change original system.

We pass the whole stripe and some useful information to FPGA. FPGA will do filter operations with the information on that stripe. Result will be returned to software after filtration. Software will use the result to do similarity search. Because the data used to do similarity search was filtered by FPGA. Unnecessary data won't be used at this time.

For each stripe, it will be read from disk and passed to FPGA through interface. Along with the stripe, some useful information will be used by FPGA as parameter. We get the filter results from FPGA and put it into memory, it will used to do other things.

Algorithm 2. Interface to access Hardware

Input: *finfo* – ORCFile which we are processing
Input: *ficon* – filter conditions
Input: *rdata* – raw data will be used by FPGA
1: **Function** visit_FPGA(*finfo, ficon, rdata*)
2: **if** we want to use FPGA to do filter job **then**
3: **for** each stripe $s \in finfo$ **do**
4: $t \Leftarrow$ coarseFil(*ficon*, *s*);
5: **if** *t* is true **then**
6: get *sinfo* from stripe;
7: get *raw_data* from stripe;
8: set_para_FPGA(*sinfo*);
9: get_result(*raw_data*);
10: put each result for stripe into memory pool;
11: **for** each key/value in memory pool **do**
12: pass down the data to do similarity search;

We get each stripe information from metadata (Line 6). Then the raw data waiting to be processed will be read from disk (Line 7). Function set_para_FPGA will be used to set parameters for FPGA (Line 8). We can get the final result from FPGA through function get_result (Line 9).

3.3 Mechanism of FPGA

In recent years, many researches use FPGA (field-programing gate arrays) to process high-volume data, e.g., data mining [10, 11], image processing [12–14], or other high-throughput applications [15, 16]. It seems that we can make use of FPGA in the field of data clean.

The core component of FPGA is a series of processing unit. Each unit can process two kinds of judge sentences.

1. Sentence like *column* θ *constants*. In this sentence θ is compare symbol, it include =, <>, >, <, >=, <=. *Column* is the column used to do compare. *Constant* represent a constant or string, it was used to be compared.
2. Sentence like column θ column. In this sentence θ is compare symbol, it include =, <>, >, <, >=, <=. Two columns will be used to do compare.

When a query comes, CPU will analyze that query and pass the parameter to FPGA. Processing unit of FPGA can calculate whether the data meet the condition based on parameter immediately. It will send signal “1” when the condition are met. Otherwise, it will send signal “0”.

4 Experiments

4.1 Framework

The algorithms introduced in Sect. 4 have been implemented in Hadoop-2.6.0. The experiments were performed on Hadoop system with one namenode and three datanode. Each node running Ubuntu 14.04 equipped with one Intel Core i5-2400 3.10 GHz quad-core processors and 8 GB DRAM.

The dataset we used in experiments is generated by TPC-H. We will use different size to test our system because of we assumed that the lager the dataset the better the performance of this system. And our FPGA will just do filter job this time, so we will just use one table in TPC-H.

The experiments will tested with and without coarse filtration in a series of size of dataset. In this way we can see the performance of coarse filtration and FPGA alone.

4.2 With FPGA Alone

Set the MapReduce run with FPGA and run test query in this model. Compare the result with the original Hadoop. Do this in different size of dataset.

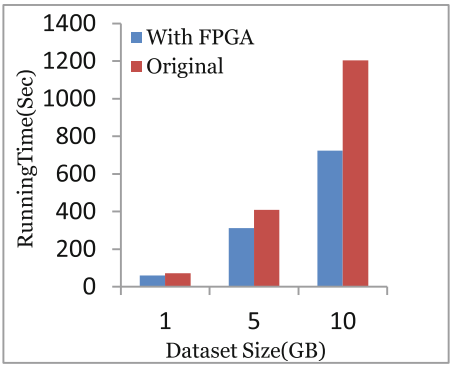

Fig. 4. Running time on three kind of dataset size

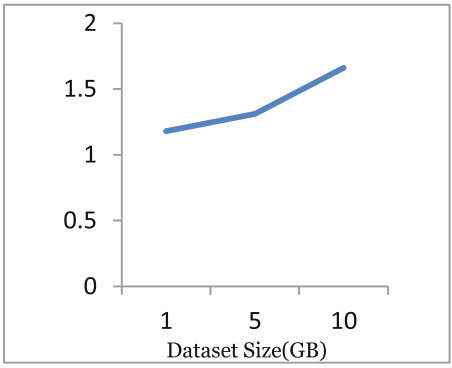

Fig. 5. Performance improved ratio

In Fig. 4 we can see two system's time cost on three different size of dataset. It shows that we can improve the performance with FPGA. Time cost by system with FPGA less than original Hadoop. Figure 5 shows the ratio between our system and original one. With the increase of the amount of data, the ratio will increase. It means our system perform better on high-volume dataset.

4.3 Add Coarse Filtration

We know how coarse filtration works from algorithm 1. It performs filtration based on statistic information of the whole stripe. So it only works on stripe rather than rows. This means the performance relate with the query sentence. And if one column of the file was ordered, the performance will be better.

In this section query sentence include a range query on the ordered column. Therefore, the coarse filtration will be used. Otherwise, we cannot test the performance of it (Figs. 6 and 7).

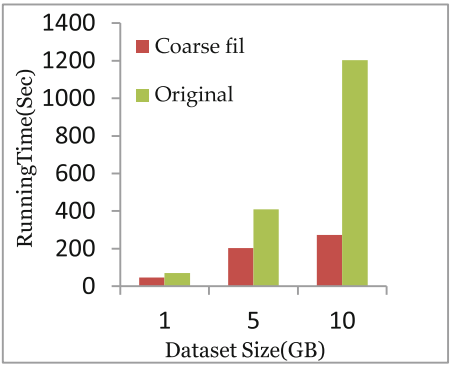

Fig. 6. Running time on three kind of dataset size

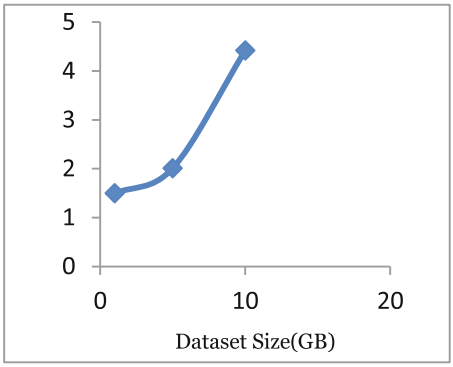

Fig. 7. Performance improved ratio

We can imagine that if the range query on ordered column is fixed, the large the dataset the more data will be filtered. The performance improved significantly because of the filtered data won't be read from disk.

However, we can't say this System is the best due to its performance rely on the query sentence. If we can't filter data from it, its performance is not better than the system with FPGA alone.

5 Conclusion and Outlook

In this paper, we proposed filtration based on FPGA to improve the performance of data clean system based on Hadoop. We want to reduce the running time of the most time-consuming part. We use FPGA to do filtration job to reduce I/O time and ease CPU's pressure. The experiment results show that our system performs better than the original one.

Coarse filtration performs better during the query sentence include range query on the ordered column. We can filter large amount of data through it when the dataset is ordered. If the dataset is disorganized, coarse filtration will not bring us benefits. In order to fix this problem we can chose use it or not based on the query sentence.

We can do many things with FPGA [17–19] because of its inherent advantages in data process. We want to implement join, group by and other operations with FPGA. We hope FPGA and other hardware play a huge role in massive data process in the future.

Acknowledgements. This paper was partially supported by National Sci-Tech Support Plan 2015BAH10F01 and NSFC grant U1509216, 61472099, 61133002.

References

1. Rahm, E., Do, H.: Data cleaning: problems and current approaches. IEEE Data Eng. Bull. **23**(4), 3–13 (2000)
2. Morales, G.D.F., Lucchese, C., Baraglia, R.: Scaling out all pairs similarity search with mapreduce. In: 8th Workshop on LargeScale Distributed System for Information Retrieval (2010)
3. Bayardo, R.J., Ma, Y., Srikant, R.: Scaling uop all pairs similarity search. In: Proceeding of WWW (2007)
4. Awekar, A., Samatova, N.F.: Fast matching for all pairs similarity search. In: Intelligent Agent Technology Workshop (2009)
5. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. In: Proceedings of the 6th OSDI, vol. 51, no. 1, pp. 107–113 (2004)
6. HDFS (Hadoop Distributed File System) Architecture. http://hadoop.apache.org/core/docs/current/hdfs_design.html
7. Sukhwani, B., Hong, M., Thoennes, M., Dube, P., lyer, B.: Database analytics acceleration using FPGAs. In: International Conference on Parallel Architectures and Compilation Techniques, pp. 411–420 (2012)
8. http://docs.hortonworks.com/HDPDocuments/HDP2/HDP-2.0.0.2/ds_Hive/orcfile.html
9. https://cwiki.apache.org/confluence/display/Hive/LanguageManual+ORC
10. Woods, L., Teubner, J., Alonso, G.: Real-time pattern matching with FPGAs. In: IEEE International Conference on Data Engineering, pp. 1292–1295 (2011)
11. Teubner, J., Muller, R., Alonso, G.: Frequent item computation on a chip. IEEE Trans. Knowl. Data Eng. **23**(8), 1169–1181 (2011)
12. Zarifi, T., Malek, M.: FPGA implementation of image processing technique for blood samples characterization. Comput. Electr. Eng. **40**(5), 1750–1757 (2014)
13. Brost, V., Yang, F., Meunier, C.: Flexible VLIW processor based on FPGA for efficient embedded real-time image processing. J. Real-Time Image Process. **9**(1), 47–59 (2014)
14. Chenini, H., Dérutin, J.P., Aufrère, R., Chapuis, R.: Parallel embedded processor architecture for FPGA-based image processing using parallel software skeletons. J. Adv. Sig. Process. **2013**(1), 1–23 (2013)
15. Choi, Y.M., So, K.H.: Map-reduce processing of K-means algorithm with FPGA-accelerated computer cluster. In: IEEE International Conference on Application-specific System, Architectures and Processors, pp. 9–16 (2014)
16. Belean, B., Borda, M., Bot, A.: FPGA based hardware architectures for iterative algorithms implementations. In: International Conference on Telecommunications and Signal Processing, pp. 751–754 (2013)

17. Becher, A., Bauer, F., Ziener, D., Teich, J.: Energy-aware SQL query acceleration through FPGA-based dynamic partial reconfiguration. In: International Conference on Field Programmable Logic and Applications, pp. 1–8 (2014)
18. Dennl, C., Ziener, D., Teich, J.: On-the-fly composition of FPGA-based SQL query accelerators using a partially reconfigurable module library. IEEE Int. Symp. Field-Programma Custom Comput. Mach. **282**(1), 45–52 (2012)
19. Halstead, R.J., Sukhwani, B., Min, H., Thoennes, M., Dube, P., Asaad, S., Iyer, B.: Accelerating join operation for relational databases with FPGAs. In: Proceeding of the 2013 IEEE 21st Annual International Symposium on Field-Programmable Custom Computing Machines, pp. 17–20 (2013)