International Journal of Hybrid Information Technology
Vol. 10, No.1 (2017), pp. 149-160
http://dx.doi.org/10.14257/ijhit.2017.10.1.13

Map-Reduce-Join-Locate: a Data Processing Framework for Decreasing the Processing Cost on Large Data

Shu-Hai Wang[1, 2], Gui-Lan Liu[1], Li-Hua Han[1] and Zhao-Hui Qi[1]

[1]*College of Information Science and Technology, Shijiazhuang Tiedao University, Shijiazhuang Hebei, China*
[2]*The Key Laboratory for Health Monitoring and Control of Large Structures, Hebei province, Shijiazhuang Hebei, China*
wsh36302@126.com

Abstract

The related retrieval operation of cloud database is often very time-consuming, takes up a lot of storage and network transmission cost. MapReduce model provides processing framework for cloud database connection operation, but the processing performance should be further optimized. Based on the analysis of the MapReduce processing framework, this paper proposes a Map-Reduce-Join-Locate processing framework. The framework consists of four phases, Map, Reduce, Join and Locate. The new framework can be deployed on the original MapReduce framework without additional modification. Experiments show that the proposed framework enhances the performance of the associated cloud database retrieval in time and space. The framework also applies to the star-connected operation of the data warehouse and the associative retrieval operation of the application secondary indexes.

Keywords*: MapReduce; Hadoop; Cloud platform; Connection; Query*

1. Introduction

With the rapid development of e-commerce [1,2], telecommunications[3], finance [4] and other applications as well as telemedicine [5], smart home [6], *etc.*, the data for supporting these applications take a rapid growth in a geometrical ratio. There exist great deals of structured and unstructured data. These data are stored by a distributed way and managed by a unified way. Cloud database for the storage massive amounts of data provides a viable platform [7,8]. Query processing for cloud data is mainly completed in the programming process framework, such as, MapReduce [9], and Dryad [10]. Although MapReduce can process data retrieval operation in parallel, it's still a consuming time and labor intensive operation in exceptionally large cloud database environment. On the one hand, MapReduce was originally designed to solve the isomorphic mapping data sets (map), aggregation (reduce). On the other hand, performing the connection operation will generate Cartesian data set and a large number of temporary intermediate results, storage and transmission. It is divided into n or $2(n-1)$) (n is the number of the connection data tables) MapReduce process. Each MapReduce needs to transmit the connecting size of the dataset, so that the more MapReduce needs, the greater amount of data transmission, and the more initialization MapReduce spending shuffling and sorting stages needed. To solve these problems, there are four major research approaches. The first is to perform a variety of connectivity algorithms on existing MapReduce framework. Second, by modifying existing MapReduce programming framework, the connection operation can be more convenient and efficient completion. The third is often performed for the data table connection to establish a connection index advance. The fourth establishes a connection view to the data table frequently performed connection.

ISSN: 1738-9968 IJHIT

In this paper, based on the suitable for connecting operations existing MapReduce framework, we expanded MapReduce processing framework further, and proposed Map-Reduce-Join-Locate processing framework. The processing framework provides an efficient solution to associated retrieval beyond the conventional MapReduce.

2. The Related Works

On the conventional MapReduce framework, the whole process is divided into two processes, Map and Reduce. In the Map phase, it extracts data by conventional filtering algorithms. In the Reduce phase, it generates needed data set through aggregation algorithm. Its data processing model is the following.

$Map: \quad (key1, value1) \rightarrow list(key2, value2)$

$Reduce: \quad (key2, list(value2)) \rightarrow list(value2)$

Overall process flow is shown in Figure 1.

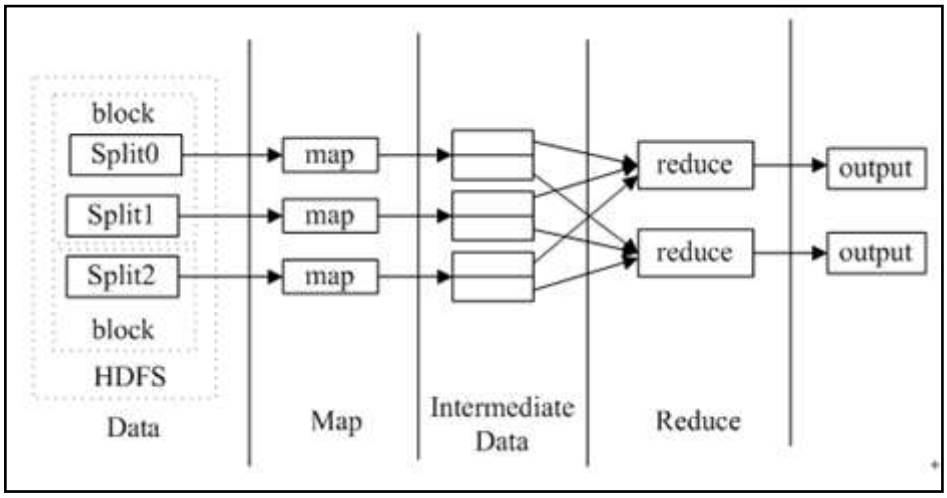

Figure 1. MapReduce Processing Model

From Figure 1, we can find four main schemes for dealing with data. The schemes are the following.

(1) The built-processing model of MapReduce is mainly used for processing homogeneous data sets by filtering - aggregating operations.

(2) The mapping results are stored on the hard disk at the form of temporary intermediate key-value pairs.

(3) Reduce phase extracts mapping results as input data to achieve aggregation of data.

(4) The results of Reduce phase are stored in HDFS (Hadoop Distributed File System).

However, if one implements directly associative retrieval of data sets in the MapReduce framework, $A \bowtie B \bowtie C$, as shown in Figure 2, he will encounter the following challenges.

(1) When dealing with multiple processes involved in MapReduce, it needs more processing time, storage space cost.

(2) The connecting results between data sets as a temporary intermediate result data are stored on the hard disk. This will lead to spend a great storage space.

(3) Each MapReduce processing results are stored in HDFS, which increases the time cost of node check.

(4) Massive amounts of data transmitted over the network will increase the network traffic and reduce network throughput.

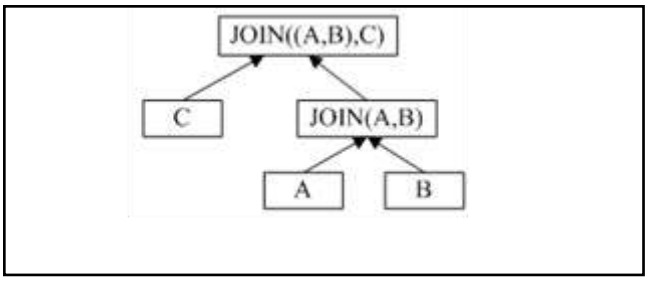

Figure 2. Example of Associative Retrieval, $A \bowtie B \bowtie C$

In order to overcome the challenges from the conventional MapReduce framework, a researcher takes lots of related study. The first scenario for the study primarily filters tuples which don't satisfy conditions. It is intended to reduce the amount of data generated when tables are on Cartesian connection. Implementation of the method is dependent on the development of the flow of program control to write the data stream realization [11,12]. The second program of study includes Map-Join-Reduce [13], Map-Reduce-Merge [14], Scatter-Gather-Merge [15], *etc*. The difference among researches is the time of executing connection. The third and fourth solution is a relational database connection reference index and view technology, respectively. Because the key-value is not compressed when storing key information view, the connection index is relatively more efficient than the view method. It only applies to the query conditions and accesses to property fixed. However, the storage cost of establishing connection index and connection view in massive cloud database will be unusually large.

3. The Proposed Map-Reduce-Join-Locate Processing Framework

3.1. The Proposed Map-Reduce-Join-Locate Model

The proposed Map-Reduce-Join-Locate is an extended MapReduce model in terms of relevance search functions. It achieves filtering, aggregation, connectivity, position location operation by dividing associative retrieval into four stages, Map, Reduce, Join and Locate.

Now, let S_A and S_B be connection data set, respectively. An attribute value of S_A corresponds to a attribute value of S_B. K_{SA} and K_{SB} represent key information data sets of S_A and S_B, respectively, without any duplicate values. V_{SA} and V_{SB} represent value information data sets of S_A and S_B, respectively. It only contains the used attribute value during the connection. *list* denotes a collection with the same key information. Then, we can describe a process of data processing, $S_A \bowtie S_B$.

Map:
$$(K_{SA1}, V_{SA1}) \rightarrow (list([K_{SA2}, 0], V_{SA2}))$$
$$(K_{SB1}, V_{SB1}) \rightarrow (list([K_{SB2}, 1], V_{SB2}))$$

Reduce:
$$\left.\begin{matrix}([K_{SA2}, 0], V_{SA2}) \\ ([K_{SB2}, 1], V_{SB2})\end{matrix}\right\} \rightarrow ([list(K_{SA3}), 0], [list(K_{SB3}), 1], V_{SA3}), \ when \ V_{SA2} = V_{SB2}$$

Join:
$$([list(K_{SA3}), 0], [list(K_{SB3}), 1], V_{SA3}), \rightarrow ([list(K_{SA3}), 0]) * [list(K_{SB3}), 1], \ V_{SA3})$$

Locate:
$$([K_{SA4}, 0], [K_{SB4}, 1], V) \ \rightarrow (V_{SA5}, \ V_{SB5}).$$

In this processing model, the *Map* function makes slice data sets as input data. It extracts *key* information needed during the connection and expresses it as *key* □ *value* pairs, based on the filter processing logic. In the *Reduce* phase, it pulls provisional list of results from the mapping results. It hashes connection property values for each data set. It gathers *key* information with the same attribute values together. They are described as $[list(key, identity), value]$. It filters out the data which do not meet the gather conditions of values or have only one identifier. Then the reatment results are delivered to the join node for joining processing. In *Join* connection phase, it pulls data from the output of *Reduce* phase and performs the *Join* operation. Finally it generates *key* □ *value* pairs, $([K_{SA3},0],[K_{SB3},1],V_{SA3})$. In *Locate* phase, it accesses data from specified node and data block according to the generated *key* information during the connection phase. Eventually it returnes to the user. Unlike the original MapReduce model, the proposed new model at its *Map* stage only extracts the needed properties of the connection process. These needed properties are described as the *key* □ *value* pairs containing source of data identification. In *Reduce* phase, it conbines the data with the same *value* and the *key* connection attribute information from different data sets to generate the *key* list. The overall implementation process example of data processing, $S_A \bowtie S_B$, is shown in Figure 3, where $S_A.F_{SA1} = S_B.F_{SB1}$.

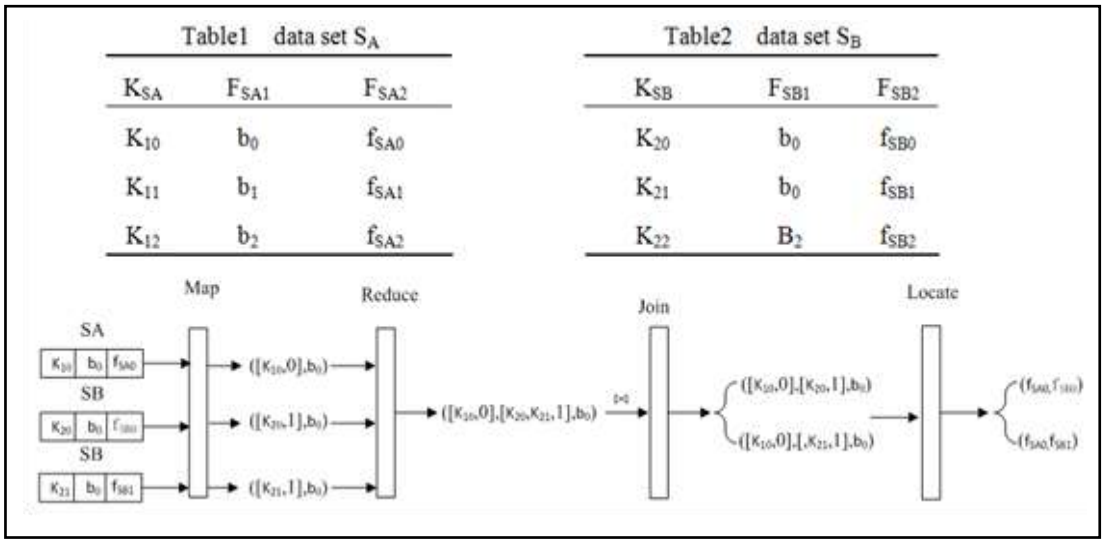

Figure 3. Data Processing of the Map-Reduce-Join-Locate Model

3.2. Associative Retrieval Algorithm

This section includes four sub-algorithms about the four stages of the proposed Map-Reduce-Join-Locate model when it performs a join query, $S_A \Theta S_B$. The four sub-algorithms show the model how to work together and give the details of each part.

Sub-algorithm 1, *Map Algorithm*
< Map Phase >
Input: *k* is a *key*, *v* is a tuple of each participating join datasets
Output: a set of *key* □ *value* pairs
Map (const Key& k, const Value& v) {

if it has constraint on dataset one or more attributes then
filtering_map(v); /* Filter data which does not satisfy the conditions of
/* single-value tuples */
end if;
Switch (v in which dataset)
Case "S_A":
Output_key = ([k,0]);
Output_value = (v_1); /* v1 is the connection property values */
Break;
Case "S_B":
Output_key = ([k ,1]);
Output_value = v_1;
Break;
Emit (Output_key, Output_value);
}

Sub-algorithm 2, *Reduce Algorithm*

< Reduce Phase >

Input: k is a key. v is a value of each participating join datasets, and (k,v) is the result of

Map

Output: a set of *key* □ *value* pairs while the keys get together with the same join attribute

Reduce (const Key& k, const Value& v){
if it has aggregating constraint on dataset then
filtering_reduce(v); /* Filter data which does not satisfy the conditions of
/* tuples aggregate functions */
end if;
if the key comes from one dataset about v then
filtering_reduce(v); /* filter out only a single source of *key* □ *value* pairs */
end if;
Emit (k, list(v));
}

Sub-algorithm 3, *Join Algorithm*

< Join Phase >

Input: k_1 / k_2 is a key towards a list with the same value from S_A / S_B, respectively. v is a

value, and $([list(k_1),0],[list(k_2),1],v)$ is the result of *Reduce* phase.

Output: a set of *key* □ *value* pairs

Join (const list<Key1>& k_1, const list<Key2>& k_2, const Value& v){
Cartesian (k_1, k_2);
Emit (list ([k_{1i}, 0], [k_{2j}, 1]));
end if;
}

Sub-algorithm 4, *Locate Algorithm*

< Locate Phase >

Input: k is the result of *Join* phase

```
Output: the final results
Locate (const Key& k) {
        ts = GetDetails (k);
        return ts;
}
```

3.3. Cost Evaluation of Model

In the implementation of associative retrieval, retrieval efficiency can be measured from both time cost and storage space occupied. The storage space can be easily solved by increasing the storage capacity of a computer. The user is primarily concerned with its waiting time. So in this paper, it considers the time cost of the Map-Reduce-Join-Locate processing model. To analyze its retrieval efficiency, the time cost can be represented by the following formula.

$$T_{total} = C_{tr}T_{traversal} + T_{storage} + T_{transmission} + C_{re}T_{results}$$

The parameters C_{tr} and C_{re} are impact factors related to hardware compute speed. The cost $T_{traversal}$ denotes the time of traversing the data set. The $T_{results}$ is the time of returning results. For the two costs, different treatment frameworks are almost consistent with one another. So, here the costs are considered as constants. The data storage time $T_{storage}$ can be measured by the I/O quantity. On the other hand, the transmission time $T_{transmission}$ is proportional to the amount of network transmission. It can be measured by the amount of the transmission data through the network considering to estimate the time. Now, let N be the amount of data. The P denotes the percentage of the storage data. Then the total time cost C_{total} can be represented as the following formula.

$$C_{total} = (C_{tr} + C_{re}P)N + C_{st}N_{I/O} + C_{trs}N_{transmission}$$

From the formula, we can find that for a fixed query, reducing the amount of data storage and data transfer amount is equivalent to the reduction of the total time of the query. Throughout the process, the *Map* phase reads the fragmented data as input and the processing results are stored into disk. At the *Reduce* phase, the results of *Map* phase are used to the input of this phase. Then the outputs of *Reduce* phase and *Join* phase are directly used to the inputs of their next phases. So in *Reduce* phase and *Join* phase, there is no writing I/O cost due to no data into the disk. In *Join* phase and *Locate* phase, there is no reading I/O cost due to no data from the disk. Only in *Locate* phase, the final results will be written to HDFS. Therefore, we can compute the cost of the proposed model as the following.

$$MapRead_{I/O} = \sum_{i=1}^{n} |SE_i|$$

SE_i, the i-th data set of the connection; $|SE_i|$ is the amount of data set i; n, the number of pieces of data.

$$MapWrite_{I/O} = ReduceRead_{I/O} = \sum_{i=1}^{n} |SE_i| * P_{mi}$$

P_{mi} is the proportion of data after filtering in the Map stage;

$$D_{reduce-join} = \sum_{i=1}^{n} KP_i * |L_i| * P_{ri}$$

KP_i, the number of key□value pairs after aggregation; $|L_i|$, the amount of data of key□value pair after gathering in Reduce phase; P_{ri}, the proportion of data after removing the data without meeting the conditions in Reduce phase;

$$D_{join\text{-}locate} = \sum_{i=1}^{n} KP_i * |Tr_i| * Pr_i$$

$|Tr_i|$, *the transmission cost of data location according to the aggregated key□value pairs;* Pr_i, *the selection probability of each data set* i.

$$LocateWrite_{I/O} = \sum_{i=1}^{n} KP_i * |LW_i| * Pr_i$$

$|LW_i|$, *the writing cost of data location to HDFS.*

So, we can get the overall estimation cost C_{total} of the proposed Map-Reduce-Join-Locate processing model.

$$C_{total} = MapRead_{I/O} + MapWrite_{I/O} + ReduceRead_{I/O} + D_{reduce-join} + D_{join\text{-}locate} + LocateWrite_{I/O}$$

$$= \sum_{i=1}^{n} |SE_i| + 2\sum_{i=1}^{n} |SE_i| * P_{mi} + \sum_{i=1}^{n} KP_i * |L_i| * P_{ri} + \sum_{i=1}^{n} KP_i * |Tr_i| * Pr_i + \sum_{i=1}^{n} KP_i * |LW_i| * Pr_i$$

For the same data set, the connection only extracts the used properties. The transmitted data of the proposed model are less than the original model MapReduce. The I/O cost of the new model is relatively small. However, the new model adds a new stage, *Locate* phase. This will bring some location cost. For a small amount of data, the proposed model has no more advantage. But, when the data amount is large, the overall data transferring cost of the original MapReduce model will be great.

4. Integrating the Map-Reduce-Join-Locate Model with MapReduce

The proposed Map-Reduce-Join-Locate model can be built on the original MapReduce framework. The original MapReduce-based programs can be easily transplanted to the Map-Reduce-Join-Locate. The *Map* phase of new model is inserted to the Map phase of MapReduce framework. Then *Reduce* phase of MapReduce is replaced by the Reduce-Join-Locate of new model. Figure 4 gives the integrated framework.

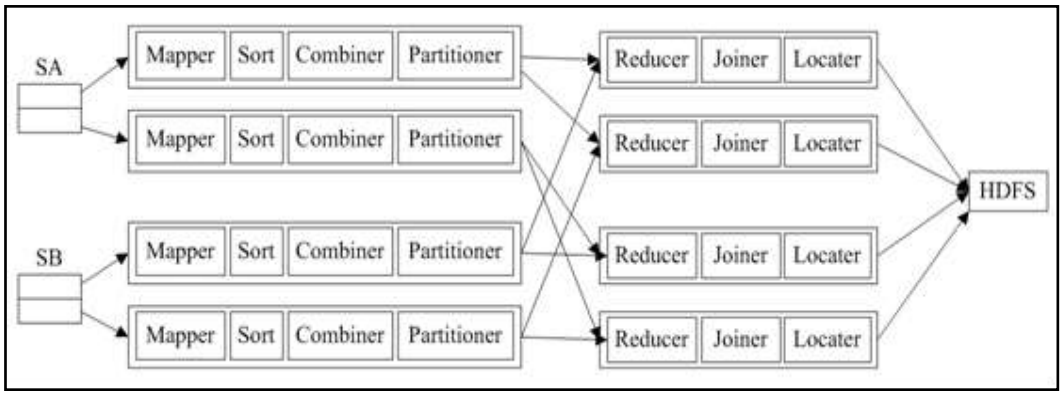

Figure 4. Workflow of the Map-Reduce-Join-Locate model

In *Map* phase, it reads data set in splits. It extracts valuable property information through a custom Mapper function, filters tuples which is not satisfied conditions. The data set is described as the form of *key□value*. The *Hash* attribute of *key□values* is used to assign them to the matched reducing nodes. Next, the Reducer function takes the output of Mapper function as input. It removes the data which does not satisfy the conditions. Then the tuples with the same connection properties are merged together. the Reducer function directly passes the merged tuples to the Joiner function for the

connection processing. Locater function pulls key information from Joiner function. Finally, it gets the needed data from the corresponding data sets based on keys and returns the final results to users.

5. Experiment and Analysis

Experiment environment is on the cloud platform, *Hadoop*. It creates two tables for connection, UserInfo and Blog. The table UserInfo records names of users. The table Blog records the blogs published by users. Here, our task is to build the connections between UserInfo and Blog. Two methods, MapReduce and Map-Reduce-Join-Locate framework, are used to finish this task. The intermediate data and the time cost for this task is used to evaluate and analyze the performance of the proposed Map-Reduce-Join-Locate model.

Experiment 1, about the comparison of intermediate data by different methods

A large number of temporary intermediate results when performing operations on the data sets will bring more storage and transmission cost. The conventional MapReduce produces many temporary intermediate results because of Cartesian operations on data sets. In *Experiment 1*, we recorded the temporary intermediate data by the two different methods, as shown in Figure 5.

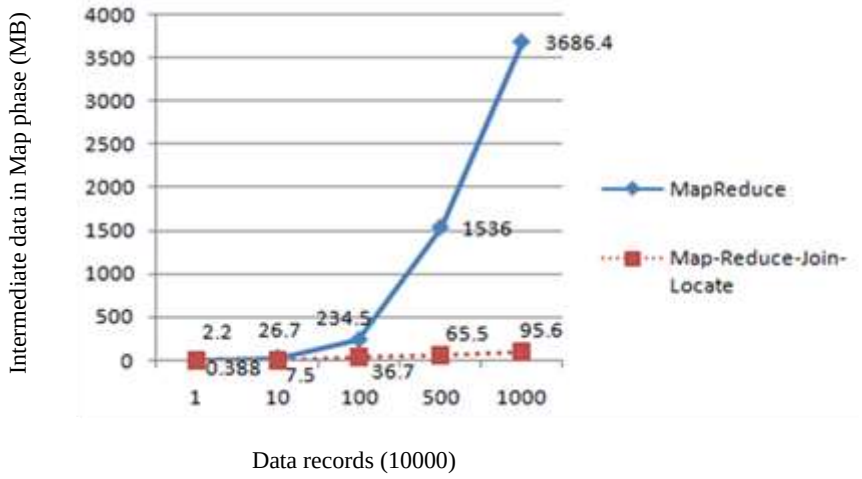

Figure 5. Comparison of Intermediate Data by Different Methods

As one can see from Figure 5, when considering a small amount of data (less than 100000 records), we can find that the intermediate data by MapReduce is approximately 10 times as much as by the proposed the Map-Reduce-Join-Locate model. With the increase of data in the database entries, in *Map* phase the intermediate data by MapReduce shows a sharp increase. However, the growth of the intermediate data by the Map-Reduce-Join-Locate model generates more gentle.

Experiment 2, about the comparison of database connections by different methods

In experiment 1, we have discussed the amount of intermediate data by two methods. The intermediate data is directly related to the cost of data storage and data transmission. Next, for simplification, in this experiment 2 we only consider the cost of database connection by different methods instead of database connection, data storage and data transmission. Figure 6 describes the time cost of database connections in two child nodes by two different methods.

From Figure 6, the time cost of the Map-Reduce-Join-Locate model is more than the MapReduce framework. This is because the Map-Reduce-Join-Locate model adds several processing steps, such as *Join* and *Locate*. At the same time, we can find that with the growth of data records in the database, the difference of the time cost by two methods is

less and less. That is to say, for database connections, the newly added processing steps in proposed model are almost negligible when data scale is quite large. Next, on the basis of experiment of Figure 6, we add a sub-node. The results are show in Figure 7. The total time cost has been reduced due to three child nodes. The change tendency of the time cost by two methods is similar to two child nodes.

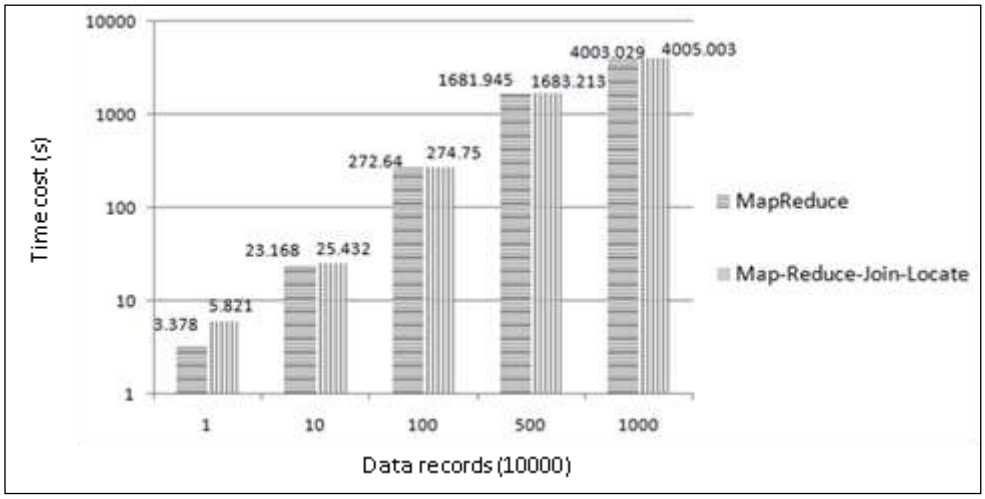

Figure 6. Time Cost of Database Connections in Two Child Nodes

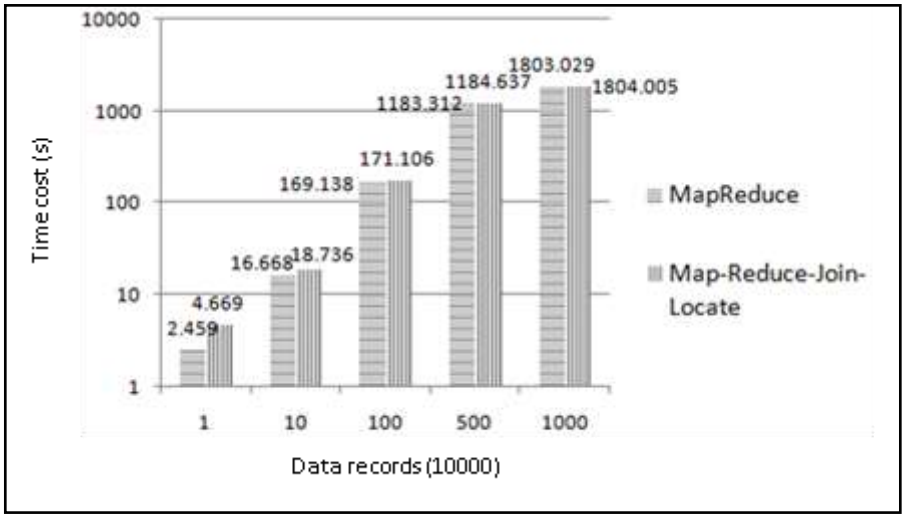

Figure 7. Time Cost of Database Connections in Three Child Nodes

In the above two experiments, we have not considered the transmission cost of data in network. It is well-known that the more the transmission data in network is, the more the time cost is. As can be seen from the experiments, the intermediate results are greatly different from each other. Obviously, the intermediate results determine the time cost of network transmission. As for the proposed model of this paper, it can obtain less time cost because of less intermediate results.

6. Conclusions

Due to limitations of MapReduce in processing connection operation, we propose a Map-Reduce-Join-Locate processing framework. Compared with the original MapReduce, the proposed Map-Reduce-Join-Locate model adds two new processing steps, *Join* and *Locate* , after implementing *Map* and *Reduce* stage. The *Join* stage

carries out the connections of data sets. The *Locate* stage takes the positioning operation of the final results. Next, the proposed Map-Reduce-Join-Locate model is deployed on the original MapReduce processing framework. Some developed programs of MapReduce is easily ported to deal with the new framework. Then, based on the cloud platform, *Hadoop*, the new model is implemented. Experiments show that the proposed framework enhances the time and space performance of the associated cloud database retrieval. The framework can also take other applications, such as the star-connected operation of the data warehouse and the associative retrieval operation of the application secondary indexes.

Acknowledgments

This work supported by the National Natural Science Foundation of China (Grant No. 61304176, 61272254), and by the Science and Technology Plan of Hebei Province, China (Project No. 15212203D)

References

[1] B. Youcef, "From e-commerce to social commerce: a framework to guide enabling cloud computing", Journal of theoretical and applied electronic commerce research, vol. 8, no. 3, **(2013)**, pp. 12-38.
[2] S. Lee and Y. Park, "The classification and strategic management of services in e-commerce: Development of service taxonomy based on customer perception", Expert Systems with Applications, vol. 36, no. 6, (**2009**), pp. 9618-9624.
[3] M. Owczarczuk, 2009, "Churn models for prepaid customers in the cellular telecommunication industry using large data marts", Expert Systems with Applications, vol. 37, no. 6, **(2009)**, pp. 4710-4712.
[4] B. Chandramouli, M. Ali, J. Goldstein, B. Sezgin and B.S. Raman, "Data stream management systems for computational finance", Computer, vol. 43, no. 12, **(2010)**, pp. 45-52.
[5] J. Clark, H. Muller and X.H. Gao, "Medical imaging and telemedicine - from medical data production, to processing, storing, and sharing: A short outlook", Computerized medical imaging and graphics, vol. 30, no. 6-7, **(2006)**, pp. 329-331.
[6] D.N. Monekosso and P. Remagnino, "Data reconciliation in a smart home sensor network", Expert systems with applications., vol. 40, no. 8, **(2013)**, pp. 3248-3255.
[7] G.C. Deka, "A survey of cloud database systems", It Professional., vol. 16, no. 2, **(2014)**, pp. 50-57.
[8] T. Ivanov, I. Petrov and A. Buchmann, 2012, "A survey on database performance in virtualized cloud environments", International Journal Of Data Warehousing And Mining, vol. 8, no. 3, **(2012)**, pp. 1-26.
[9] J. Dean and S. Ghemawat, "MapReduce: a flexible data processing tool", Communications of the ACM, vol. 53, no. 1, **(2010)**, pp. 72-77.
[10] M. Isard, M. Budiu and Y. Yu, "Dryad: distributed data-parallel programs from sequential building blocks", ACM SIGOPS Operating Systems Review, vol. 41, no. 3, **(2007)**, pp 59-72.
[11] K.S. Beyer, V. Ercegovac and R. Gemulla, 2011, "Jaql: A scripting language for large scale semistructured data analysis", Proceedings of the 37th International Conference on Very Large Data Bases, Washington, USA, **(2011)**.
[12] S. Blanas, J.M. Patel and V. Ercegovac, "A comparison of join algorithms for log processing in mapreduce", Proceedings of the 2010 ACM SIGMOD International Conference on Management of data, New York, USA, **(2010)**.
[13] D. Jiang, A. Tung and G. Chen, "MAP-JOIN-REDUCE: Toward scalable and efficient data analysis on large clusters", IEEE Transactions on Knowledge and Data Engineering,. vol. 23, no. 9, **(2011)**, pp. 1299-1311.
[14] H. Yang, A. Dasdan and R.L. Hsiao, 2007, "Map-reduce-merge: simplified relational data processing on large clusters", Proceedings of the 2007 ACM SIGMOD international conference on Management of data, Beijing, China, **(2007)**.
[15] H. Han, H. Jung and H. Eom, 2011, "Scatter-Gather-Merge: An efficient star-join query processing algorithm for data-parallel frameworks", Cluster Computing, vol. 14, no. 2, **(2011)**, pp. 183-197.

Authors

Shu-Hai Wang, he is a professor in College of Information Science and Technology at Shijiazhuang Tiedao University. He received the PhD in computer engineering from Tianjin University, China, in 2010. His research interests in Big data and Computer Information System.

Gui-Lan Liu, she is a student in College of Information Science and Technology at Shijiazhuang Tiedao University. Her research interests in big data and software developing.

Li-hua Han, she is an associate professor in College of Information Science and Technology at Shijiazhuang Tiedao University. She received the Master of education technology from Beijing Jiaotong University, China, in 2007. Her research interests in Information System and Network education.

Zhao-Hui Qi, he is a professor in College of Information Science and Technology at Shijiazhuang Tiedao University. He received the MS and the PhD in computer science from Tianjin University, China, in 2003 and 2006. His research interests in Bioinformatics and Pattern Recognition.

