The Optimization of Cost-Model for Join Operator on Spark SQL Platform

Xin Lian^{1,2,*}, and Tianyu Zhang^{1,2}

¹Chongqing University of Posts and Telecommunications, College of Computer Science and Technology, 400065 Chongqing, China

²Chongqing Engineering Research Center of Mobile Internet Data Application, 400065 Chongqing, China

Abstract. Spark needs to use lots of memory resources, network resources and disk I/O resources when Spark SQL execute Join operation. The Join operation will greatly affect the performance of Spark SQL. How to improve the Join operation performance become an urgent problem. Spark SQL use Catalyst as query optimizer in the latest release. Catalyst query optimizer both implement the rule-based optimize strategy (RBO) and cost-based optimize strategy (CBO). There are some problems with the Catalyst CBO module. In the first place, the characteristic of In-memory computing in Spark was not fully considered. In the second place, the cost estimation of network transfer and disk I/O is insufficient. To solve these problems and improve the performance of Spark SQL. In this study, we proposed a cost estimation model for Join operator which take the cost from four aspects: time complexity, space complexity, network transfer and disk I/O. Then, the most cost-efficiency plan could be selected by using hierarchical analysis method from the equivalence physical plans which generated by Spark SQL. The experimental results show that the total amount of network transmission is reduced and the usage of processor is increased. Thus the performance of Spark SQL has improved.

1 Introduction

Spark SQL analyze SQL and generate execution plans by using Catalyst query optimizer. A SQL Statement will be analyzed from an abstract syntax tree (AST) to a logical plan by Catalyst and the logical plan will be optimized by Catalyst at same time. Then the logical plan will be analyzed by Catalyst to the physical plan. Finally, the physical plan will be submitted to the Spark platform for execution. Like AST, both logical plan and physical plan are tree structure [1].

The Join operator is one of the most complex operators in Spark SQL. In the Catalyst physical plan, the Join operator have multiple implementation ways, just such as Shuffle-Hash-Join, Broadcast-Hash-Join and Sort-Merge-Join. In big data environment, if a table size is very small, the optimal way of performing a join operation is Broadcast-Hash-Join undoubtedly, because of its highest efficiency. But once the data has increased tremendously, the usages of resources about memory and bandwidth for broadcasting will inevitably become larger, so that Broadcast-Hash-Join would not be the best choice.

Currently, Spark SQL optimization strategy is mainly divided into two major directions: rule-based optimization (RBO) and cost-based optimization (CBO). Rule-based optimization is an empirical, heuristic optimization approach, which relies on the optimization rules which has been summarized by expert before. Rule-based optimization is useful for simple operators,

but it is hard to optimize join operator by using rules. Using cost-based optimization, we first need to get statistics information about the corresponding tables and columns. By using these statistics information, we can estimate the cost of different operators in one SQL statement and then select the optimal execution plan according to the estimation result. In the cost-based optimization method, the most important factor is the cost evaluation model and the execution plan selection model.

In Spark 2.2 release, a cost-based optimization strategy was implemented. It also implemented a SQL reordering mechanism. However, its cost estimation model is needed to be improved, especially on the complex operators such as Join operator.

In recent research work of Spark SQL Cost-Based optimization, the characteristic of In-memory computing in Spark has not fully considered. In some operations, Spark will load too much data into the memory which will overwriting the data to the disk and trigger JVM garbage collect frequently.

The crucial contributions are as follows:

1. A Comprehensive Model on cost estimation in time/space complexity and I/O cost aspects was proposed. And the cost estimation models for each kind of Join Operator implementations were proposed.
2. A physical plan selection strategy based on the cost estimation model was proposed.

* Corresponding author: xlianxin@163.org

2 Related Works

In Spark SQL platform, a SQL statement would be translated by Catalyst analyzer to a logical plan. Then the logical plan would be translated by Catalyst analyzer to multiple physical plans which were equivalent.

By adding cost model, the cost of each operator in the physical plan will be estimated, and then among the multiple plans, a physical plan with the lowest overall cost will be selected.

Chunlei Liu proposed a cost evaluation model for different operators in paper [2]. By using the obtained operator cost, a greedy algorithm based multi-table join order selection method has also been proposed. The paper [3] proposed the Spark Bloom optimization based on Partial Bloom Filter for large-scale equivalent join operation. This method can both better filter out the records of the table which do not meet the conditions of the join keys and significantly reduce the amount of data transferred at the shuffle stage, which had effectively improved the performance of large table equivalent join operation. Paper [4] proposes another algorithm of BloomFilter Re-partition, uses the algorithm to filter out most of the invalid connections that do not meet the criteria.

Paper [5] used a kind of hybrid storage medium which gave full play to the advantages of each storage medium. And the author proposes a cost estimation model for hybrid storage that automatically chooses valuable data to cache which will improve system query processing performance. Paper [6] after deeply study the Spark SQL workflow, a storage middle layer was added between the underlying persistent file system and the Spark core to solve the problem of sharing the data between queries. It reduced disk I/O overhead, reduced memory usage and improved performance of Spark SQL.

Paper [7] combined the advantages of Simi-Join and Partition Join and provides an optimized equi-join algorithm based on the features of Spark. Paper [8] used ant colony algorithm to optimize the query plan in distributed database, and simulated it in Oracle database to verify the effectiveness and practical significance of the model.

This study analyzed the existing optimization strategy for Spark SQL at physical plan phase and analyzed the Spark SQL query optimization mechanism which based on RBO and CBO. The query optimization mechanism based on CBO will improve the query optimizer capabilities and improve the overall performance of Spark SQL.

3 Related Definitions

To calculate the cost of each Spark SQL physical plan, we need to build a cost model for each operator in the physical plan of the tree structure. In the cost model, the mainly calculation object is the related table in the database and the types of operator.

Table 1 shows the symbols used in the cost model and their definitions.

Table 1. Symbols and Definitions.

Symbol	Definition
O_{type}	Time complexity
S_{type}	Space complexity
P_{type}	Network transmission cost
D_{type}	Disk I/O cost
K	Row selectivity, default value is 0.2
T	Related table
$ T $	The number of rows in the related table
T_L	The left table
$ T_L $	The number of rows in the left table
T_R	The right table
$ T_R $	The number of rows in the right table
N	The number of the Nodes
$Buffer_{shuffle}$	The buffer size of the shuffle operator
$tupleSize$	The average size of the tuple, in bytes
$Max()$	The maximum function, return the maximum value of the parameters

As shown in table 1, the $|T|$ represent the number of the rows in the related table, which can get by Spark SQL API. The K represent the selectivity of the rows means there are about 20% rows will match join condition and we set its default value at 0.2. The O_{type} represent time complexity cost, the S_{type} represent the space complexity cost, the P_{type} represent the network cost, the D_{type} represent the disk I/O cost and its subscript type means the type of SQL Operation. And there are several types in all: Shuffle, HashJoin, BJoin (Broadcast-Hash-Join), SHJoin (Shuffle-Hash-Join), SMJoin (Sort-Merge-Join), BNLJoin (Broadcast-Nested-Loop-Join).

4 Cost Model of Join Operators

The cost estimation model in this study is the overall cost of one physical plan, which include time complexity cost, space complexity cost, network transfer cost and disk I/O cost. Due to different cluster has its different performance of network and disk, the cost of the disk I/O and network should take consider separately. Such as some cluster will use 1GbE network, some cluster will use 10GbE network, some cluster will use traditional mechanical hard drivers and some cluster will use solid-state drivers as storage.

4.1 Cost Model of Basic Operators

In order to calculate the cost of Join operator, we have to estimate the cost of the Shuffle operator and the cost of the HashJoin operator firstly.

4.1.1 Cost Model of Shuffle Operator

Shuffle operator require a large amounts of data transfer operations. In Spark Platform, the Shuffle operation was divided into the Map phase and the Reduce phase.

In the Map phase, the Mapper nodes will create a batch of buckets. And the number of buckets corresponding to the number of Reducer nodes and the number of Mapper nodes. The number of buckets is $M \times R$, where M represent the number of Map nodes, and R represent the number of Reduce nodes.

In the Reduce phase, Spark will distribute the desired Map results to the Reduce nodes.

Therefore, the Shuffle operator's time complexity cost, space complexity cost, network cost and disk I/O cost could be obtained by formula (1), formula (2), formula (3) and formula (4):

$$O_{\text{shuffle}}(T) = |T| \times \log(|T|) \quad (1)$$

$$S_{\text{shuffle}}(T) = \text{Buffer}_{\text{shuffle}} \quad (2)$$

$$P_{\text{shuffle}}(T) = |T| \times \text{tupleSize} \quad (3)$$

$$D_{\text{shuffle}}(T) = \text{Max}(|T| \times \text{tupleSize} - \text{Buffer}_{\text{shuffle}} \times N, 0) + |T| \times \text{tupleSize} \quad (4)$$

4.1.2 Cost Model of HashJoin Operator

When Spark execute HashJoin operation, it will scan the table twice, with the time complexity of $O(|T_L| + |T_R|)$. HashJoin has significantly improved the performance than Join operations using Cartesian products. However, HashJoin consumes more memory because of HashJoin need to create a hash table in memory. Meanwhile, HashJoin operator only applies to the equal-join operation.

Therefore, the HashJoin operator's time complexity cost, space complexity cost could be obtained by formula (5) and formula (6):

$$O_{\text{HashJoin}}(T_L, T_R) = \frac{|T_L| + |T_R|}{N} \quad (5)$$

$$S_{\text{HashJoin}}(T_L, T_R) = \frac{|T_L| + |T_R|}{N} \quad (6)$$

4.2 Cost Model of Join Operators

Currently, Spark SQL supports four types of join operation: Shuffle-Hash-Join, Broadcast-Hash-Join, Sort-Merge-Join, and Broadcast-Nested-Loop-Join. Shuffle-Hash-Join relies on Spark Shuffle and HashJoin operations, Broadcast-Hash-Join only relies on HashJoin operation, Sort-Merge-Join is a sort-based join operation, Broadcast-Nested-Loop-Join is a Join operation that applies to LeftJoin, RightJoin and OuterJoin.

4.2.1 Cost Model of Broadcast-Hash-Join

Spark will use the broadcast variables for broadcasting the smaller tables to the Map nodes, then shuffle the larger tables and complete the HashJoin in the Map node.

Therefore, the Broadcast-Hash-Join operator's time complexity cost, space complexity cost, network cost and disk I/O cost could be obtained by formula (7), formula (8), formula (9) and formula (10):

$$O_{\text{BJoin}}(T_L, T_R) = O_{\text{HashJoin}}(T_L, T_R) \quad (7)$$

$$S_{\text{BJoin}}(T_L, T_R) = S_{\text{HashJoin}}(T_L, T_R) + |T_L| \quad (8)$$

$$P_{\text{BJoin}}(T_L, T_R) = N \times K \times |T_L| \quad (9)$$

$$D_{\text{BJoin}}(T_L, T_R) = D_{\text{shuffle}}(T_R) \quad (10)$$

4.2.2 Cost Model of Shuffle-Hash-Join

When execute Shuffle-Hash-Join, Spark will firstly shuffle the two tables. The data with the same Join Key will distributed into the same node. Then, Spark execute HashJoin operations within the nodes to take full advantage of cluster performance. Shuffle-Hash-Join operation can be divided into two stage, as shown in Figure 1:

Fig. 1. The Procedure of Shuffle-Hash-Join

The first stage: Spark shuffle two tables to ensure that the data with the same Join Key will distributed into the same node. The two tables will be evenly distributed to each node in Spark.

The second stage: The HashJoin stage will execute HashJoin operations in each node.

Therefore, the Shuffle-Hash-Join operator's time complexity cost, space complexity cost, network cost and disk I/O cost could be obtained by formula (11), formula (12), formula (13) and formula (14):

$$O_{\text{SHJoin}}(T_L, T_R) = O_{\text{shuffle}}(T_L) + O_{\text{shuffle}}(T_R) + O_{\text{HashJoin}}(T_L, T_R) \quad (11)$$

$$S_{\text{SHJoin}}(T_L, T_R) = S_{\text{shuffle}}(T_L) + S_{\text{shuffle}}(T_R) + S_{\text{HashJoin}}(T_L, T_R) \quad (12)$$

$$P_{\text{SHJoin}}(T_L, T_R) = K \times |T_L| \times N + K \times |T_R| \times N \quad (13)$$

$$D_{\text{SHJoin}} = D_{\text{shuffle}}(T_L) + D_{\text{shuffle}}(T_R) \quad (14)$$

4.2.3 Cost Model of Sort-Merge-Join

Spark SQL will use Sort-Merge-Join for Join operations when cluster memory capacity is limited. Sort-Merge-Join operation was divided into three stages:

The first stage: Similar to the first stage of Shuffle-Hash-Join, it will shuffle two tables to ensure that the data with the same Join Key will be distributed into the same node. The two tables will be evenly distributed to each node in Spark.

The second stage: Spark will sort the data after Shuffle within the node.

The third stage: Spark traverses two sequential data alternately and outputs the result with the same Join Key.

The procedure of Sort-Merge-Join as shown in Figure 2:

Fig. 2. The Procedure of Sort-Merge-Join

Therefore, the Shuffle-Hash-Join operator's time complexity cost, space complexity cost, network cost and disk I/O cost could be obtained by formula (15), formula (16), formula (17) and formula (18):

$$O_{SMJoin}(T_L, T_R) = O_{shuffle}(T_L) + O_{shuffle}(T_R) + \frac{|T_L| + |T_R|}{N} \quad (15)$$

$$S_{SMJoin}(T_L, T_R) = S_{shuffle}(T_L) + S_{shuffle}(T_R) \quad (16)$$

$$P_{SMJoin}(T_L, T_R) = P_{shuffle}(T_L) + P_{shuffle}(T_R) \quad (17)$$

$$D_{SMJoin}(T_L, T_R) = D_{shuffle}(T_L) + D_{shuffle}(T_R) \quad (18)$$

4.2.4 Cost Model of Broadcast-Nested-Loop-Join

Different from the first three types of Join operations, HashJoin is primarily for InnerJoin, while Broadcast-Nested-Loop-Join is mainly for LeftJoin, RightJoin and OuterJoin. Broadcast-Nested-Loop-Join also relies on broadcast variables, but it does not rely on Shuffle operation and HashJoin operation. Its implementation based on loop traversal.

Therefore, the Broadcast-Nested-Loop-Join operator's time complexity cost, space complexity cost, network cost and disk I/O cost could be obtained by formula (19), formula (20), formula (21) and formula (22):

$$O_{BNHJoin}(T_L, T_R) = |T_L| \times |T_R| \quad (19)$$

$$S_{BNHJoin}(T_L, T_R) = |T_L| + S_{shuffle}(T_L) \quad (20)$$

$$P_{BNHJoin}(T_L, T_R) = K \times |T_L| \times N + P_{shuffle}(T_R) \quad (21)$$

$$D_{BNHJoin}(T_L, T_R) = D_{shuffle}(T_R) \quad (22)$$

5 The Physical Plan Selection Strategy Based on Cost Model

5.1 Cost Estimation Method using AHP

Using the cost model, we can obtain the time complexity cost, space complexity cost, disk I/O cost and network transfer cost of different implementations of Join operator. In order to select the most cost-effective physical plan among the multiple physical plans which generated by Spark SQL, we should take advantage of the cost analysis of the Join operation in different dimensions and find a suitable mathematical model to get the weighted cost.

$$\text{Cost}(T) = f(O(T), S(T), P(T), D(T)) \quad (23)$$

In this study, we use AHP model to help us choose the most cost-efficiency physical plan. We select time complexity cost, space complexity cost, disk I/O cost and network transfer cost as criteria for evaluating the physical plan alternatives. The overall design of AHP as shown in Figure 3:

Fig. 3. Model of AHP

Firstly, to build the AHP model, the pairwise comparison matrix A should be constructed. And the matrix should be assigned at scale 1-9 which referred from the proposal of Thomas L. Saaty [9]. This mean in the matrix a_{ij} takes values from 1-9 or its reciprocal. The a_{ij} is represent the importance of dimension i compared to dimension j . Then we can use the summation method to approximate the eigenvectors U_A of the matrix A by formula (24).

$$U = (u_1, u_2, \dots, u_n) \quad (24)$$

$$u_k = \frac{\sum_{j=1}^n a_{kj}}{\sum_{i=1}^n \sum_{j=1}^n a_{ij}} \quad (25)$$

We need to sum the cost in each dimensions of each physical plan $L_1, L_2, L_3, \dots, L_n$. Then the cost

comparison matrix of each dimension should be constructed. For example, the time complexity cost

comparison matrix is $X_o = \begin{bmatrix} x_{11}, x_{12}, \dots, x_{1j} \\ x_{21}, x_{22}, \dots, x_{2j} \\ \vdots \\ x_{i1}, x_{i2}, \dots, x_{ij} \end{bmatrix}$, in the

matrix x_{ij} represent the comparison value of physical plan i with physical plan j in time complexity dimension. Similarly, the comparison matrix X_s in space complexity cost dimension, the comparison matrix X_p in network cost dimension and the comparison matrix X_d in disk I/O cost dimension all could be constructed. Then, we can get the eigenvectors U_o of the time complexity cost comparison matrix according to the formula (26), get the eigenvectors U_s of the space complexity cost comparison matrix according to the formula (27), get the eigenvectors U_p of the network cost comparison matrix according to the formula (28) and get the eigenvectors U_d of the disk I/O cost comparison matrix according to the formula (29).

$$U_o = (u_{o1}, u_{o2}, \dots, u_{on}) \quad (26)$$

$$U_s = (u_{s1}, u_{s2}, \dots, u_{sn}) \quad (27)$$

$$U_p = (u_{p1}, u_{p2}, \dots, u_{pn}) \quad (28)$$

$$U_d = (u_{d1}, u_{d2}, \dots, u_{dn}) \quad (29)$$

The u_{on} , u_{sn} , u_{pn} and u_{dn} correspond to the eigenvalues of the time cost, space cost, network cost, and disk cost in the eigenvector of the related comparison matrix.

The weighted cost of each physical plan $W_L = (w_{L1}, w_{L2}, w_{L3}, \dots, w_{Ln})$ can be get by bringing the eigenvectors U_A, U_o, U_s, U_p, U_d into the formula (30). The w_{L_i} represent the weighted cost of the physical plan L_i in four aspects.

$$w_{L_i} = \sum_{j=1}^n u_{oj} a_{ij} + \sum_{j=1}^n u_{sj} a_{ij} + \sum_{j=1}^n u_{pj} a_{ij} + \sum_{j=1}^n u_{dj} a_{ij} \quad (30)$$

The lowest cost physical plan can be selected after calculate the weighted cost of physical plan L_i by using formula (31):

$$L_i = \min(w_{L1}, w_{L2}, w_{L3}, \dots, w_{Ln}) \quad (31)$$

Finally, the most cost-efficiency physical plan will submitted to the spark for execution.

5.2 Optimal Physical Plan Choice Algorithm

In Spark SQL platform, A SQL statement will generate one logical plan by using Catalyst Query Optimizer. And

one logical plan can be translated to multiple equivalent physical plans. By choose the most cost-efficiency physical plan, it will reduced the network transmission, disk I/O operation and improve the utilization of the system. Finally, it will improve the performance of Spark SQL.

Here are the algorithmic description of how to use the cost estimation model and AHP to select the model cost efficient physical plan from multiple equivalent physical plans. The algorithmic description is as shown in table 2:

Table 2. Optimal Physical Plan Choice Algorithm

Input: A set of equivalent physical plans
Output: The most cost-efficiency physical plan
Description:
(1) Traversing the nodes of each physical plan tree. According to the type of operator, we can get the cost in four aspects by using the cost model (formula 1-22). So, we can get the total cost of each plan tree in four aspects.
(2) Based on experience, we can build a pairwise comparison matrix between time, space, network, and disk cost. And get its eigenvector by using formula (25).
(3) Using the cost result, we can build the cost contrast matrix between different physical plans on the time, space, network and disk dimensions. And using formula (25), we can get the corresponding eigenvectors U_o, U_s, U_p, U_d
(4) According to the AHP model, we can use formula (30) to calculate the weighted sum of eigenvectors. So, we can get the weighted cost of each physical plan.
(5) Finally, choosing most cost-efficiency physical plan by formula (31).

6 Experimental Results

Based on Spark 2.2 release, the default cost implementation was replaced by the cost model proposed in this paper.

In the article of Liu Chunlei of UESTC, TPC-H standard benchmark was used as the experimental method. TPC-H is standard SQL benchmark which supported for business intelligence-driven decision and developed by the American Standard Trading Processing Performance Council. As the paper [10] described, TPC-H dataset compared with TPC-DS [11] dataset, its tables has featureless shortcoming. For example, TPC-H dataset lack of data skew features. And TPC-H database maintenance function did not reflect the actual performance of DBMS, like the performance of ETL (Extract-Transform-Load), its maintenance function only restrict the potential overuse of index. Because of TPC-H can no longer accurately reflect the true performance of todays distributed database systems, TPC has introduced a new generation of TPC-DS benchmarks for decision-making applications.

This paper will choose TPC-DS as benchmark dataset. TPC-DS using star, snowflake and other multi-dimensional data model. It contains seven fact tables with an average of 17 latitude tables containing 18 columns per table. This test set contains complex applications such as statistics, online queries, data

mining, and more for big dataset. The test data and values are also skewed to match the real data.

A cluster with 6 nodes was built for experiment. In the cluster, each nodes has 8 cores processor and 16G memory and 2TB Hard HDD without raid.

Experiment 1:

The comparison of performance between Spark SQL with the default cost model and Spark SQL with the cost model proposed by this paper. We choose Query04, Query11, Query25, Query85 in TCP-DS as workload and use TPC-DS Tools generate 200G datasets. Each query will run by five time on Spark Platform and take average execution time.

Fig. 4. Comparison of Performance

The experimental results was shown in Figure 4. The Default column represent the performance of Spark SQL without cost-based optimization. The CBO column represent the performance of Spark SQL with default cost model. The CBO_Mod column represent the performance of Spark SQL with replaced cost model. The result shows that after using the cost model, Spark SQL performance has improved by 12% to 74%. Meanwhile, the Spark SQL with the cost model proposed in this paper has 5% to 10% performance improvement compared with the default cost model in the condition of one SQL with complex Join operations.

Experiment 2:

Analysis the changing of performance with number of Spark worker nodes has increased. In this experiment, the size of dataset is set to 200G and choose Query05, Query13, Query25, and Query45 as workload. And the number of Worker is set to 2 to 6. In this experiment, we will compare the performance of default Spark SQL with the Spark SQL has replaced cost model. The results was shown in Figure 5,6,7,8.

The cost model is designed to reduce IO costs because the more nodes the more data that needs to be transfer during execution. The experimental results show that with the increase of cluster size, the performance gap between Spark SQL with cost model and Spark SQL without cost model is gradually bigger.

Fig. 5. Query05

Fig. 6. Query13

Fig. 8. Query25

Fig. 9. Query45

Experiment 3:

Analysis the Spark SQL resource utilization when adding cost model. Recording the resource utilization of each node during Spark SQL execution to further analyze the resource utilization. The results is shown in Table 3 and Table 4.

Table 3. The Comparison of Network

Network	Default	CBO_Mod	Diff
Send	19922.63MB	17949.44MB	1973.19MB
Receive	7113.73MB	2715.42MB	4398.31MB

Table 4. The Comparison of CPU Load

CPU Load	Default	CBO_Mod	Diff
----------	---------	---------	------

IO Wait	0.14	0.12	0.02
Load AVG	1.28	1.98	0.70

It can be clearly seen that after using the cost model, the network sending is reduced by 1900 MB, and the network receiving is reduced by 4300 MB. The CPU average load has increased which means using CPU more efficient instead of waiting for data to be transmitted over the network.

As shown in experiments, when a SQL statement has complex Join operation, by choosing the most cost-efficiency physical plan, it can both reduce the disk I/O operation and network data transmission between worker nodes. Finally, improving the overall efficiency of the cluster.

7 Conclusions

By analyzing the implementation of Query Optimizer and Join operator in Spark SQL, the execution cost of a SQL statement was evaluated in four aspects. Different cost estimation model has proposed for different Join operation implementations. And we replace Spark SQL default cost model with new cost model. The new cost model has better performance than default cost model in the SQL with complex Join operations.

By implementing and replacing the cost estimation model for Join-optimized in Spark SQL. In the SQL statement with complex join operation which involve a large amount of data transmission, the Spark SQL with cost model proposed in this paper has 5% -10% performance improvement compared with the Spark SQL with default cost estimation model, thus verifying the new cost model.

References

1. M. Armbrust, R. S. Xin, C. Lian, et al. Spark sql: Relational data processing in spark. ACM SIGMOD International Conference on Management of Data, 1383-1394 (2015).
2. C. L. Liu. Research on Cost-based Query Optimization for Spark SQL. Doctoral Thesis. University of Electronic Science and Technology of China, Chengdu, China (2016).
3. L. Zhang. Research on query analysis and Optimization Based on Spark system. Doctoral Thesis. Beijing Jiaotong University, Beijing, China (2016).
4. R. C. Liu, M. Q. Zhou, X. J. Pi, et al. Optimization of the Equi-Join Problem Based on Big Data in Spark. Modern Computer, **8**, 3-6 (2017).
5. D. Zeng. Spark SQL Data Reuse Mechanism Based on Revenue Model. Doctoral Thesis. University of Chinese Academy of Sciences, Beijing, China (2016).
6. K. Z. Cai. Query Optimization in SQL to Spark. Doctoral Thesis. Southeast University, Nanjing, China (2016).
7. H. Q. Bian, Y. G. Chen, X. Y. Du, et al. Equi-join optimization on spark. Journal of East China Normal University, **5**, 261-270 (2014).
8. F. F. Cui, Z. Q. Nan. Research on Query Optimization Based on Ant Colony algorithm in Distributed Database. Computer Era, **5**, 47-49 (2014).
9. T. L. Saaty. Analytic Hierarchy Process. Encyclopedia of operations research and management science, 64-72 (2013).
10. D. Chen, X. J. Ye, L. Shi. Implementation of performance testing for TPC-DS benchmark. Journal of Computer Applications, **9**, 2449-2452 (2011).
11. TPC-DS. <http://www.tpc.org/tpcds/>