Research on Query Analysis and Optimization Based on Spark

Yan Li
State Key Laboratory of Networking and Switching Technology
Beijing University of Posts and Telecommunications
Beijing, China
Email: zenithward@bupt.edu.cn

Hongbo Wang
State Key Laboratory of Networking and Switching Technology
Beijing University of Posts and Telecommunications
Beijing, China
Email: hbwang@bupt.edu.cn

Yangyang Li
Innovation Center
Academy of Electronics and Information Technology
Beijing,China
Email: yli@csdslab.net

***Abstract*—With the rapid development of the Internet and the explosive growth of information, the traditional technical framework can not meet the needs of massive data processing. In this environment, the research and development of big data platform came into being. Compared to Hadoop MapReduce programming model, the Spark computing framework has a better applicability by introducing RDD (Elastic Distributed Data Set) and memory-based computing model. SparkSQL is an api that integrates relational processing and Sparks functional programming. It provides a better choice for handing massive structured data. However, for the most complex and costly inter-table correlation queries in traditional query, Spark SQL's performance is poor. To some extent, it has affected the application of Spark. This paper first introduces the technical background of Spark architecture, Optimizer Catalyst, and then expounds the factors causing low query performance. Then, a design scheme of cost optimization and predicate pushdown is proposed based on Spark SQL. The proposed scheme is based on scalable Catalyst, which improves the performance degradation due to improper selection of table association algorithm and the triggering of shuffle. Finally, the Spark cluster test environment is built to verify the feasibility and performance improvement of the proposed scheme.**

***Index Terms*—Big data , Spark SQL , Catalyst , cost optimization, hash join**

I. Introduction

With the continuous development of the Internet industry and the popularity of all kinds of Internet applications, a large amount of structured data will be generated every day in many fields, such as telecommunications, transportation, finance, etc. As the amount of data continues to grow, traditional data storage (DBMS) and computing methods (stand-alone programs) have failed to meet the needs of enterprises for statistical analysis and knowledge mining. In big data environment, the performance requirements of query analysis data continue to improve, and the traditional database such as MySQL provides massive data storage scheme, but still has obvious bottleneck in query. MySQL can only use one CPU core per query, whereas Spark can use all cores on all cluster nodes[1].

The arrival of big data age[2], also guide to big data processing platform of the ecosystem has been constantly updated. With the birth of HDFS, MapReduce, HBase, Hive and other Hadoop ecosystems, Hadoop[3] has become the preferred platform for big data processing. Based on Google's extensive papers[4-5] and experience on massive data processing, Hadoop has implemented algorithms and set up the stack to make large-scale batch processing easier to use. MapReduce relies heavily on persistent storage. Each task needs to perform read and write operations multiple times, so it is relatively slow. Thus a new big data processing engine Spark[6] came into being. Spark, developed with Hadoop's MapReduce engine based on the same principles, focuses on speeding up the speed of batch workloads through improved memory and processing optimization mechanisms. This article is based on spark big data platform for query optimization.

In the mass of structured data set to do SQL. Hadoop uses its own hive query engine in the early days. However, Hive has a fatal flaw: it is based on MapReduce, and MapReduce's shuffle is disk based, thus causing Hive's performance to be abnormally low. Later, Spark launched Shark, Shark and Hive are actually closely related. Shark built on a lot of things that rely on Hive, but modified the memory management, physical planning and the implementation of three modules. It uses memory-based computing model of Spark in the bottom. So that the performance is better than Hive several times to hundreds of times.However, Shark still has its problem, Shark depends on the Hive's syntax parser, query optimizer and other components. Therefore, the improvement of its performance is still a constraint. Since then the Spark team decided to completely abandon Shark, launched a new project named Spark SQL. Spark SQL plays an important role in the Spark ecosystem. Spark SQL[7] makes it easy to manipulate structured data, run SQL queries in the Spark system, and easy to locate the response data tables and metadata.

II. Related Work

A. *Spark SQL runtime architecture*

Figure 1 shows the runtime architecture of SparkSQL. The order of execution of the SparkSQL statement is: 1. Parse the SQL statement, distinguish the SQL statement into the keywords(such as SELECT, FROM, WHERE), expression, Data Source, etc., to determine whether the SQL statement conforms to the specification; 2. Bind the SQL statement and the database data dictionary(column,table,views), if the relevant Projection, Data Source, etc., are present. It means that the SQL statement can be executed; 3. The general database will provide several execution plans, which typically have operational statistics, and the database will select an optimal plan in these plans; 4. Schedule Execute, in the order of Operation,Data Source,Result.

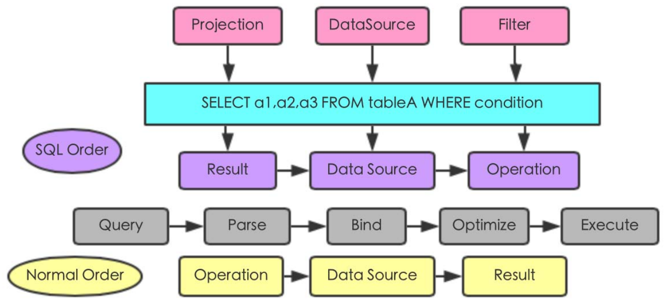


Fig. 1. SparkSQL runtime architecture

B. Catalyst

In order to make Spark SQL better, Spark platform developers have designed a new extensible optimizer Catalyst[8]. Catalyst is easy to add new optimization techniques and functions in Spark SQL to accommodate different data analysis. Catalyst supports both rule-based and price-based optimizations.

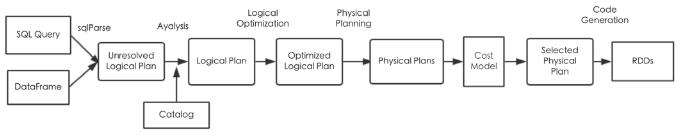


Fig. 2. Catalyst optimization process

As shown in Figure 2, the basic implementation of these components: Firstly, generate the tree by analyzing SQL statement. Then, different rules are applied to the tree at different stages, and the fuctions of each component are completed by transformation. Secondly, ayalyzer uses analysis rules to imporve the properties of Unresolved Logical Plan and transform into Resolved Logical Plan with metadata (such as hive metastore, schema catalog). Thirdly, optimizer uses Optimization Rules to convert Resolved Logical Plan to Optimized Logical Plan by optimizing operations such as merging, column clipping and push down.

C. The reason of query speed slower

Currently SparkSQL supports three Join algorithms - broadcast hash join, shuffle hash join and sort merge join [9]. The first two belong to the hash join. But it needs to shuffle or broadcast before hash join.

In the join algorithms described above, Broadcast Join has higher efficiency when large tables are connected to small table. If you broadcast large table to all nodes on the Spark cluster, the network communication overhead and disk I/O overhead can be enormous. This leads to poor performance of Join operations in this case. Therefore, broadcast Join has higher efficiency for broadcasting small tables, but it is inefficient to do association operations between two large tables. As for the Shuffle join algorithm, the most important reason for the speed reduction is that all the key/value needs to be repartitioned according to the key. This step involves the Shuffle operations. And the amount of data in the shuffle operation phase is larger. When the amount of data in the Shuffle phase is large, the network communication overhead and disk I/O overhead will be great, which will affect the performance of the entire association operation directly. Currently, the association between large tables catalyst defaults to using the Sort-Merge join algorithm. Careful analysis shows that the cost is not smaller than Shuffle hash join, but much more. The reason for selecting it is related to the shuffle implementation of Spark, which is currently applicable to the Shuffle implementation of the Sort-Merge Join algorithm for Spark. Thus, after shuffle, the partition data is sorted by key. Therefore, in theory, you can expect that the data does not need sort after shuffle, and it can merge directly.

There are some studies in academia. Which join algorithm should be chosen to perform, BroadcastJoin or ShuffleHashJoin or SortMergeJoin? Zhang Lei proposed a optimization scheme based on Part Bloomfilter algorithm in a large table related . Its liitation is the association of large tables only. In real life, the design of reasonable database tables and the design of SQL statements rarely appear in lage table association scenarios[9]; Liu Chunlei proposed an optimization model based on cost model [8], which is not perfect for connection support.

D. Contribution of the this paper

In table association queries, Different implementation strategies have different resource requirements, and the efficiency of implementation is different from each other. In the face of the same SQL, it may take a few seconds to choose the right execution strategy, and if you don't select the right execution strategy, you might cause the system out of Memory. Different Join sequences mean different execution efficiencies. The way the current SparkSQL selects the join algorithm is based on manually setting parameters. If the amount of data in a table is smaller than this value, the Broadcast Hash Join will be used, but this scheme is not elegant and not flexible. The cost optimization proposed in this paper is aimed at solving such problems. The design principle of the specific scheme will be described in detail on the following sections.

III. Optimization Design For Table Association

The cost optimization stragegy is mainly select a minimum cost syntax tree from multiple possible syntax trees. To evaluate the cost of a given whole tree, it is necessary to evaluate the cost of each node. Finally accumulate all the nodes cost. The SQL syntax tree is shown below.

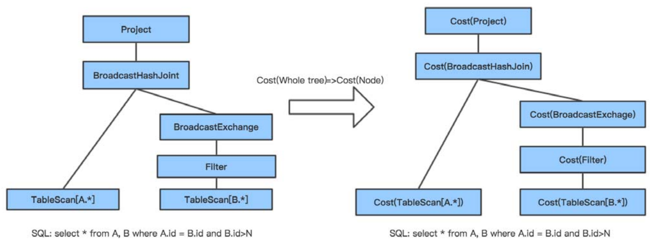


Fig. 3. Catalyst optimization process

A. *Get the basic information*

EstimatedSize: the size of each LogicalPlan node output data (decompression)

RowCount: The total number of output data for each LogicalPlan node

BasicStats: Basic column information, including column type, Max, Min, number of nulls, numberof distinct valuesmax column lengthaverage column length etc..

Histograms: Histograms of columns, i.e.,equi-width histogram(for numeric and string type) and equi-height histogram (only for numeric types)

EstimatedSize and RowCount mainly to evaluate the execution cost of actual operator. BasicStats and Histograms are used to obtain the basic information of raw data, so as to facilitate subsequent deduction.

B. *Define the cardinal derivation rules*

Based on the current statistics of the nodes, a set of inference rules for calculating parent node related statistics is defined. For different operators, the rules must be derived from the same. Such as filter, group by, limit, and so on. Evaluation of the derivation is different. The query analysis discussed in this article is mainly for filter.For such a SQL: select * from A, B where A.id = B.id and B.id >N, the syntax tree is as Figure 4.

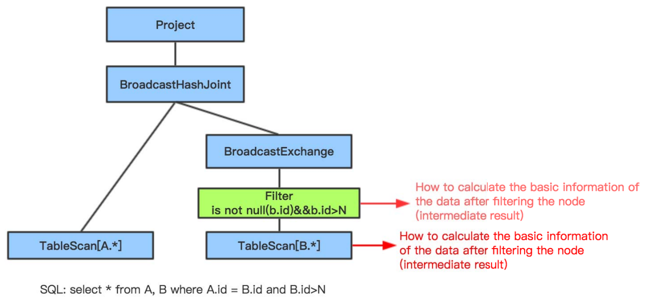

Project
BroadcastHashJoint
BroadcastExchange
Filter
is not null(b.id)&&b.id>N
TableScan[A.*]
TableScan[B.*]
How to calculate the basic information of the data after filtering the node (intermediate result)
How to calculate the basic information of the data after filtering the node (intermediate result)
SQL: select * from A, B where A.id = B.id and B.id>N

Fig. 4. Catalyst optimization process

Suppose the minimum value of the B column is B.id.Min, the maximum value is B.id.Max, and the total number of rows is B.id.Distinct. We assume that the data are evenly distributed.

There are three cases. The first case, N is smaller than B.id.Min. The second case, N is greater than B.id.Max. And the third case, N is between B.id.Max and B.id.Max. The first two cases are special cases of the third cases, as shown in Figure 5.

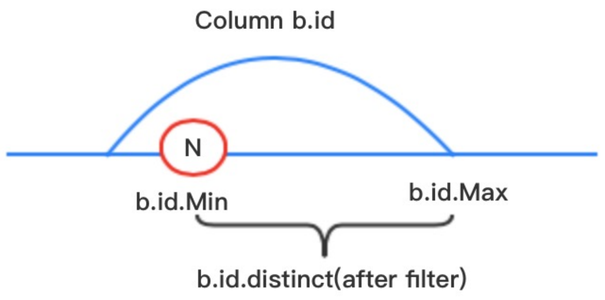


Fig. 5.

Under the filter condition of B.id bigger than N, the B.id.Min will increase to N, and the B.id.Max remains unchanged. Total number of rows after filtration is as follows:

$$B.id.distinct(after, filter) = (B.id.Max - N)/(B.id.Max - B.id.Min) * B.id.distinct(before, filter) \quad (1)$$

Actually, the distribution of data is almost impossible to be balanced. The data are usually a probability distribution, as shown in Figure 6:

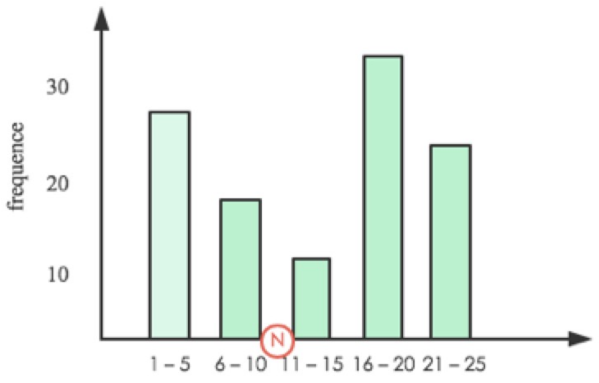


Fig. 6.

Suppose the location of the N, like the picture above. The total number of rows after filtration is as follows:

$$B.id.distinct(afterfilter) = height(> N)/height(All) * B.id.distinct(beforefilter) \quad (2)$$

By evaluating the rules and the statistics of the original table, you can count the basic statistics of all the intermediate nodes in the syntax tree.

C. The actual cost of computing core operators

The actual cost of nodes is defined from two dimensions: CPU Cost and IO Cost. Define some basic parameters as shown in TABLE 1:

TABLE I
SOME BASIC PARAMETERS

Hr	The cost of reading 1byte data from HDFS
Hw	The cost of writing 1byte data to HDFS
Tr	the number of tuples in the relation
Tsz	Average size of the tuple in the relation
CPUc	CPU cost for a comparison in nano seconds
Net	the average cost of transferring 1 byte over network in the Hadoop cluster from any node to any node

1) Table Scan operator: Scan operators are generally located in the leaves of the syntax tree, and intuitively speaking, this type of operator is only IO Cost, and CPU Cost is 0.

$$TableScanCost = IOCost = Tr * Tsz * Hr \tag{3}$$

2) Hash Join operator: Take the Broadcast Hash join as an example, assuming that the large tables are distributed on n nodes (R1,R2,...,Rn), and the average number of data in each node is Tr (R1) Tsz (R1), Tr (R2) Tsz (R2), ... ,Tr (Rn) Tsz (Rn), small table data bits Tr (Rsmall) Tsz (Rsmall), then the CPU cost and IO cost are:

$$\begin{aligned} CPUCost = {} & SmallTableBuildHashTableCost + \\ & LargeTableDetectionCost = \\ & Tr(Rsmall) * CPUc + \\ & (Tr(R1) + Tr(R2) + ... + Tr(Rn)) * N * CPUc, \end{aligned} \tag{4}$$

$$\begin{aligned} IOCost = {} & SmallTableScanCost + \\ & Smalltablebroadcastcost + LargeTableScanCost \\ & = TrRsmall * Tsz(Rsmall) * Hr + \\ & n * Tr(Rsmall) * Tsz(Rsmall) * Net + \\ & (Tr(R1) * Tsz(R2) + ... + Tr(Rn) * Tsz(Rn)) * Hr \end{aligned} \tag{5}$$

At this point, the actual cost of any node can be assessed.

D. Select the cost minimum execution path

Depending on the different join algorithms, we can get different SQL execution path, use dynamic programming to select the optimal cost path, the algorithm is as follows:

Since the weight of the path is not negative and the shortest path of the single source is obtained, we choose the Dijkstra algorithm. Initially, S contains only the starting point. Suppose that u is a vertex of G, the path from start point to u and in the middle only through the vertex of S is called the special path from start point to u. And use the array dist to record the shortest special path length corresponding to each vertex. The Dijkstra algorithm extracts the vertex u with the shortest special path length from V-S, then adds u to S. Make necessary modifications to array dist. Once S contains all the vertices in the V, dist records the shortest path length from the source to all the other vertices.

Algorithm 1 DIJKSTRA

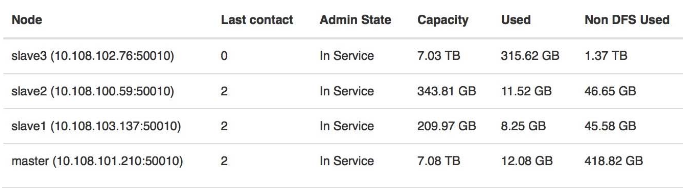
```
1: DIJKSTRA(G,w,s)
2: NITIALIZE-SINGLE-SOURCE(G,s)
3: $S \leftarrow \emptyset$
4: $Q \leftarrow V[G]$
5: while $Q \neq \emptyset$
6:     do $u \leftarrow EXTRACT - MIN(Q)$
7:         $S \leftarrow S \cup u$
8:         for each vertex $v \in Adj[u]$
9:             do RELAX(u,v,w)
```

IV. Scheme Verification

The Spark platform in this paper is built on the Hadoop environment, using the Hadoop platform HDFS as Spark Platform file system. The Hadoop platform adopts a fully distributed mode. Spark has diversified modes of operation. When it is deployed on a single machine, it can run either in local mode or in pseudo distributed mode.When the distributed cluster is deployed, there are a number of modes of operation to choose. It is established in the Hadoop system, and the management and scheduling of resources in Hadoop is manipulated by Yarn, so choose to use Yarn Cluster mode as the operation mode of Spark platform, Spark platform deployment test version is 2.0.2.

A. Testing environment

The test environment was built by four servers in the laboratory. The machine configuration is shown in the table:



Node	Last contact	Admin State	Capacity	Used	Non DFS Used
slave3 (10.108.102.76:50010)	0	In Service	7.03 TB	315.62 GB	1.37 TB
slave2 (10.108.100.59:50010)	2	In Service	343.81 GB	11.52 GB	46.65 GB
slave1 (10.108.103.137:50010)	2	In Service	209.97 GB	8.25 GB	45.58 GB
master (10.108.101.210:50010)	2	In Service	7.08 TB	12.08 GB	418.82 GB

Fig. 7. Device information

There are four test hosts that make up the test cluster for Hadoop and Spark.

B. Test cases and results

The data used in this experiment are derived from vehicle traffic structure vehicle information one hundred million mass data:

The query time statistics is base on whether SparkSQL uses cost optimization in association query. We use some different

Query to test the performance, and the test result is shown in Figure 8: The result shows that the query time is little

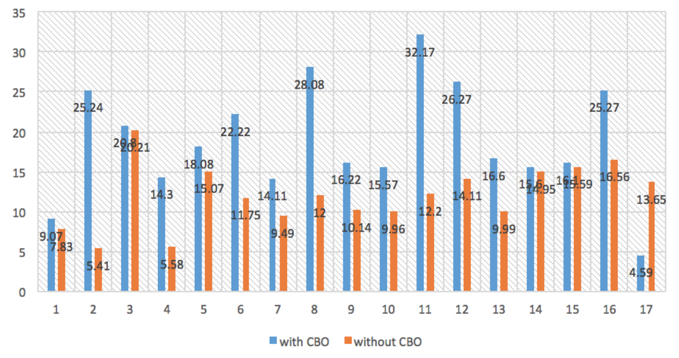


Fig. 8. Table information

different for some Query statements before and after cost optimization, while other Query statements have significant effect on cost optimization. These remarkable queries have some similar features: filter can filter out a lot of data. SparkSQL selects Shuffle Hash Join based on the size of the raw data. But Broadcast Join might be a better choice based on cost optimization. The same SQL test statement is given as follows :

```
val qualifiedRows = jdbcDF.groupBy(col("
    motor_no").as("grp_id")).count().
    filter(s"count>4")
val joinedRows = jdbcDF.join(
    qualifiedRows,jdbcDF.col("motor_no")==
    =qualifiedRows.col("grp_id")).select(
    jdbcDF("*"),qualifiedRows("count"))
```

Compare query time of different data sizes. As shown in the following figure, there were three kinds of query time. The first one is mysql inquiries, the second one is SparkSQL inquiries and the SparkSQL CBO is the optimization scheme proposed in this paper.

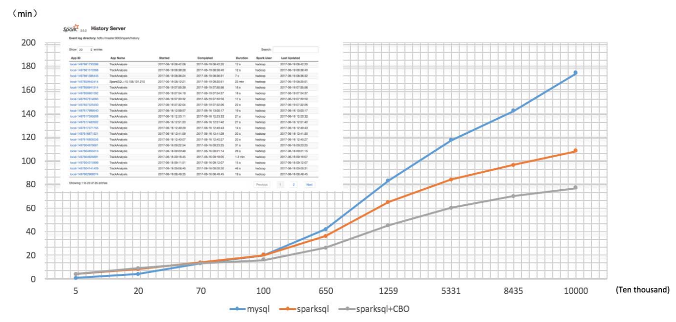


Fig. 9. Table information

This experiment shows the time of running the same SQL statement on different amounts of data. Compare the query times of MySQL, Spark SQL, and SparkSQL with cost optimization. When the data set is small, MySQL has good performance. But as the amount of data continues to increase, the query time of MySQL is close to exponential growth. SparkSQL does not work as well as MySQL in small data set. But with the increasing amount of data, the performance of SparkSQL is not as good as the optimized one. On large data sets, the optimization scheme of this paper has better effect, it takes less time to query. From the above two experiments, the feasibility and performance of this design are proved.

V. Conclusion

In this paper, we propose a cost based optimization scheme for Spark SQL to improve the performance of data query. Based on different sql statements and different sizes of test data. It can achieve very good optimization results when catalyst can filter more data and not join between tow large table. The cost based optimization scheme implemented in this paper has limitations, and the efficiency is not obviously improved when the amount of data is too large. It improve the performance by up to 4 or 5 times. And the actual throughput is also increases.

Acknowledgment

This work is supported by the National Natural Science Foundation of China (No.61002011); the 863 Program of China (No.2013AA013303); the Fundamental Research Funds for the Central Universities (No.2013RC1104)

References

[1] Floratou A, Minhas U F, zcan F. Sql-on-hadoop: Full circle back to shared-nothing database architectures[J]. Proceedings of the VLDB Endowment, 2014, 7(12): 1295-130.
[2] Manyika J, Chui M, Brown B, et al. Big Data: The Next Frontier For Innovation, Competition, And Productivity[J]. Analytics, 2011.
[3] White T. Hadoop: The definitive guide[M]. " O'Reilly Media, Inc.", 2012.
[4] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters[C]// Conference on Symposium on Opearting Systems Design and Implementation. USENIX Association, 2004:10-10.
[5] Chang F, Dean J, Ghemawat S, et al. Bigtable: a distributed storage system for structured data[C]// Symposium on Operating Systems Design and Implementation. USENIX Association, 2006:205-218.
[6] Armbrust M, Zaharia M, Das T, et al. Scaling spark in the real world[J]. Proceedings of the Vldb Endowment, 2015, 8(12):1840-1843.
[7] Park K, Peng L. A Design of High-speed Big Data Query Processing System for Social Data Analysis: Using Spark SQL[J]. International Journal of Applied Engineering Research, 2016, 11(14): 8221-8225.
[8] Meng X, Meng X, Meng X, et al. Spark SQL: Relational Data Processing in Spark[C]// ACM SIGMOD International Conference on Management of Data. ACM, 2015:1383-1394.
[9] Zhang Lei. Research on query analysis and Optimization Based on Spark system, [D]., Beijing Jiaotong University, 2016.
[10] LiuChunlei.ResearchonSparkSQLqueryoptimizationbasedoncost model [D]. University of Electronic Science and technology of China, 2016.
[11] Abouzied A, Abadi D J, Silberschatz A. Invisible loading:access-driven data transfer from raw files into database systems[C]// International Conference on Extending Database Technology. 2013:1-10.
[12] Cai Kaizhen. SQL to SPARK query optimization mechanism, [D]., Southeast University, 2016.
[13] Armbrust M, Huai Y, Liang C, et al. Deep dive into spark sqls catalyst optimizer[J]. 2015.
[14] Myalapalli V K, Savarapu P R. High performance SQL[C]//India Conference (INDICON), 2014 Annual IEEE. IEEE, 2014: 1-6.
[15] Shoro A G, Soomro T R. Big data analysis: Apache spark perspective[J]. Global Journal of Computer Science and Technology, 2015, 15(1).