Adaptive Fusion SQL Engine on Hadoop Framework

Shu-Ming Chang[1], Chia-Hung Lu[1], Jiazheng Zhou[1],
Wenguang Chen[2], Ching-Hsien Hsu[3], Hung-Chang Hsiao[4], and Yeh-Ching Chung[1]

[1]Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan, R.O.C.
[2]Department of Computer Science and Technology, Tsinghua University, Beijing, China
[3]Department of Computer Science and Information Engineering, Chung Hua University, Taiwan
[4]Department of Computer Science and Information Engineering, National Cheng Kung University, Taiwan

{shuming, lu1227, jzhou}@sslab.cs.nthu.edu.tw,
cwg@tsinghua.edu.cn, chh@chu.edu.tw, hchsiao@csie.ncku.edu.tw, ychung@cs.nthu.edu.tw

Abstract - *As big data becomes popular, data warehouse needs the ability to process massive data fast. Hive is a data warehouse built on Hadoop. It provides SQL-like query language called HiveQL. Although there is a fault tolerance mechanism in Hive, its response time is long. Impala is another SQL engine on Hadoop that is compatible with Hive. Impala's response time is short, but the data processed by Impala is constrained by memory size. Furthermore, it does not provide fault tolerance mechanism. We focus on designing a fusion SQL engine system that combines Hive and Impala. It provides a uniform interface and a fault tolerance mechanism. After the system parses the query from users, it leverages the preprocessed statistics to estimate the memory consumption, and chooses the SQL engine (Hive or Impala) to execute the query automatically. It makes Hive and Impala become complement of each other.*

Keywords: Data Warehouse; SQL Engine on Hadoop; Hive; Impala, Fusion SQL Engine

1 Introduction

Data warehouse is a database designed for reporting, data analysis and decision-making. It is usually updated in batch and it contains large amount of historical data. Apache Hadoop [1] is a platform for the distributed processing of large data sets across a cluster of commodity computers. The data warehouse for big data processing is suitable to be built on Hadoop.

Apache Hive [18] is the first data warehouse software built on Hadoop. It supports SQL-like query language called HiveQL. The HiveQL query will be compiled to several MapReduce jobs to execute. Hive relies on MapReduce, it provides fault tolerance mechanism, but the job will be run for a long time with high latency. Cloudera Impala [2] is a SQL engine on Hadoop and has its own massively parallel processing (MPP) engine. There is no disk writing in Impala except insertion operations; it processes data in memory. The characteristic makes it much faster than Hive. However, Impala does not spill to disk if intermediate results exceed the memory reserved by Impala on a node. The query would fail when any node involved processing the query is out of memory. The problem commonly occurs when joining two large tables.

In our work, we attempt to integrate Impala and Hive into a fusion SQL engine system and accomplish the followings:

- Offer a uniform query interface for Impala and Hive.
- Propose a method to estimate Impala memory usage.
- Automatic failover.
- Better response time.

The rest of the paper is organized as follows. Related work will be described in Section 2. Section 3 introduces the whole system overview and architecture. The estimation method of the query size is presented in Section 4. In Section 5, we show how to estimate Impala memory consumption. Experimental results are shown in Section 6. Section 7 are the conclusions and future work.

2 Related Work

Apache Hadoop [1] is an open-source implementation of Google File System [9] and MapReduce [7]. The Hadoop Distributed File System (HDFS) [17] provides high throughput access to data and is appropriate for large data sets. Apache Hive [18] is the first approach to data warehousing on Hadoop framework. Hive stores data on HDFS and it supports queries in SQL-like query language called HiveQL.

Shark [19] runs on Apache Spark [20] instead of MapReduce, which is a data-parallel execution engine that is fast and fault-tolerant. Cloudera Impala [2] inspired by Dremel [12] has its own massively parallel processing

(MPP) SQL query engine that runs natively in Apache Hadoop.

Ganapathi, et al. [8] propose a prediction framework that guides system design and deployment decisions such as scale, scheduling, and capacity. Gruenheid, et al. [10] use column statistics to improve the performance of HiveQL queries execution in Hive. Shi, et al. [16] propose HEDC++, a histogram estimator for data in the cloud. Okcan and Riedewald [13] focus on processing theta-joins using MapReduce. Hive [4], Shark [11] and Impala are also developing cost based optimizer to enhance the query performance.

There is less work on hybrid SQL engine architecture on Hadoop framework. Similar approach for improving data warehouse on Hadoop is HadoopDB [5]. It combines parallel database and MapReduce framework, and queries multiple single-node databases by using MapReduce as communication layer. To the best of our knowledge, we are the first system to provide fusion SQL engine system on Hadoop framework.

3 Fusion SQL Engine System

The fusion SQL engine system is designed for leveraging two different SQL engines (Hive and Impala) on Hadoop platform to make the average query response time shorter.

3.1 Overview

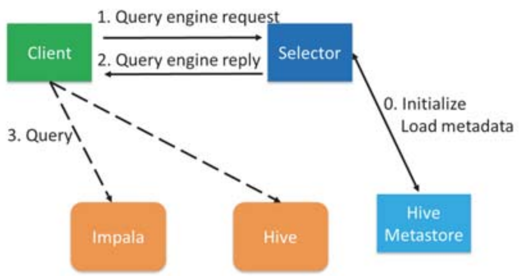

Figure 1: The Workflow of Fusion SQL Engine

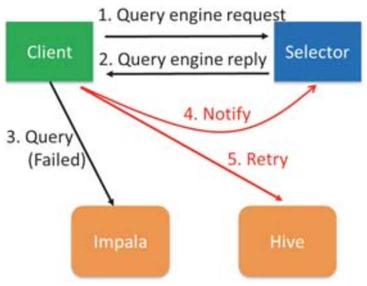

Figure 2: Failover Mechanism

Our system consists of two different SQL engines: Impala and Hive. We will get better performance when submit query to Impala, but query may fail if there is no sufficient memory. Hive can guarantee the execution will always complete but the performance is not as good as Impala. The fusion SQL engine system helps users to automatically determine which SQL engine to use when query arrives.

As shown in Figure 1 and Figure 2, when the system starts, Selector loads the metadata and statistics from Hive Metastore. Selector will use the information later for SQL engine selection decision. When Client gets a query request, Client first asks the Selector which SQL engine should be chosen. The Selector estimates the memory usage of this query and replies to Client. Moreover, our system also supports failover mechanism. As shown in Figure 2, if Impala is first selected as SQL engine, once the query fails, Client will help to perform failover mechanism. It will notify the Selector there is an estimation error and send the query to Hive to get the query result. The Selector logs every failed query in Metastore to prevent from making the wrong estimation again.

3.2 System Architecture

The software stack and architecture are shown in Figure 3. We will introduce them as follows.

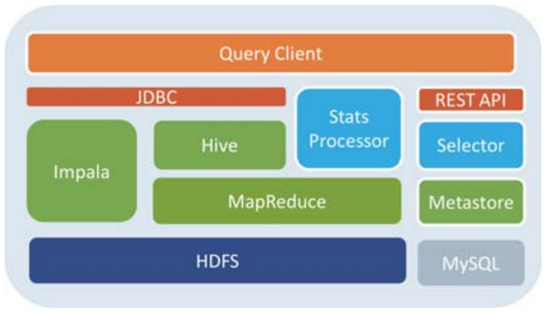

Figure 3: Fusion SQL Engine System Architecture

3.2.1 Client

The Client provides a uniform interface upon different SQL engines. It makes users use different SQL engines to execute query with the same data storage stored in HDFS.

3.2.2 Selector

Selector server loads the metadata and statistics from Hive Metastore when the system starts. Selector will use the information later for SQL engine selection decision. The information is stored in memory for fast access.

Selector estimates memory usage of a query according the execution plan from Impala. Therefore, Selector gets Impala’s execution plan in text format via JDBC, and parses it to compute the estimated memory consumption. We will explain the estimation method in Section 4.4.

There is a failover mechanism in Selector. Once the wrong estimation causes "memory limit exceeded" error, Client will notify the Selector to log this query. If Selector gets the same query next time, it will choose Hive directly instead of Impala to avoid "memory limit exceeded" error.

3.2.3 SQL Engine

There are two SQL engines in the proposed system: Impala and Hive. Impala is faster since it puts data in memory, but it lacks of fault tolerance mechanism. Compared with Impala, Hive runs slower but is reliable. Our proposed system helps users to identify the characteristics of a query by estimating memory consumption.

3.2.4 Metastore

Metastore is one of the components in Hive and used for storing metadata. It uses relational database management system to store information. The first SQL-on-Hadoop is Hive. Most of following SQL engines on Hadoop are compatible with Hive. Metadata contain table schema, partition information, table statistics and column statistics.

3.2.5 Stats Processor

Stats Processor builds statistics data that will be used for Selector. For processing the massive dataset, it is written in MapReduce to speed up the performance. This analysis step does not need to be executed every time after the system starts. Only for the first time users import the data into the data warehouse, the system executes the analysis program to collect some statistics. These statistics data can help to improve the accuracy of estimation.

4 Estimation of the Query Size

In this system, we calculate the approximate memory consumption of a query relying on estimation of query size.

4.1 Preliminary

4.1.1 Terms

- *Predicate*: Each WHERE or JOIN clause is called a predicate.
- *Cardinality*: The number of rows that is expected to return by an operation.
- *Selectivity:* A value between 0 and 1 that indicates the proportion of rows retrieved by one or multiple predicates.

When we estimate the cardinality of this SELECT operation, we need to calculate the selectivity of each predicate denoted as $SEL(P)$ where P is a predicate. We have equation (1) to calculate number of rows.

$$\#Row = Cardinality * Selectivity \qquad (1)$$

The selectivity of a predicate affects the result row size. Two methods of calculating selectivity will be described in Section 4.2 and Section 4.3.

4.2 Number of Distinct Values (NDV)

Impala uses the number of distinct values (NDV) to estimate the selectivity of a predicate in current version. The method is proposed by literature [15] and used in System R [6]. It is based on linear and uniform distribution assumption that assumes the number of occurrences of any value in a domain of an attribute is the same. The following are the selectivity formulas based on above assumption for two predicates $p = x$ and $p < x$, p and x denote an attribute and a constant of an attribute respectively.

We have equation (2) and equation (3) to calculate the selectivity.

$$SEL(p = x) = \frac{1}{NDV} \qquad (2)$$

$$SEL(p < x) = \frac{x - minval}{maxval - minval} \qquad (3)$$

NDV is the number of distinct values of attribute p, and *maxval* and *minval* are the maximum and minimum values of attribute p.

In many cases, the distribution is neither uniform nor linear. The above formulas would be inaccurate. Thus, for numeric data types, we will use histogram to improve the estimation. For non-numeric data type like string, we will use above NDV method.

4.3 Histogram

A histogram can describe the distribution of attribute values. We can build a single-dimensional histogram to record the value distribution for a single column.

4.3.1 Dimension

Single-dimensional histogram considers the frequency distribution of an individual attribute, and it is based on the attribute independence assumption that there is no correlation among the attributes. Multi-dimensional histogram considers the joint frequency distribution of several attributes. Although multi-dimensional histogram can handle complex join predicate more accurately, the overhead of building it is exponential to the number of columns. In most commercial databases, the single-dimensional histograms are often used.

4.3.2 Equi-width Histogram and Equi-height Histogram

For a traditional equi-width histogram, the internal length of each bucket is the same but the total frequency of each bucket is different. In contrast, for the equi-height histogram, the length of each bucket may be different but the total frequency of each bucket is the same. When the data are skewed, the selectivity estimation from equi-width histogram may be far from real selectivity.

As shown in Figure 4 and Figure 5, we show an example for the equi-width and equi-height histogram, respectively. In this example, most of the customers are 23 years old. If we use equi-width histogram to estimate

$SEL(age < 22)$, the error rate of estimation would be larger than that of using equi-height histogram.

The equi-height histogram is proposed by literature [14]. We leverage Piatetsky-Shapiro and Connell's proposed method to build equi-height histograms for every numeric column type and use it to estimate the query result size.

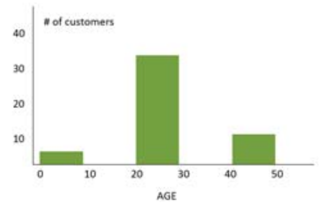

Figure 4: Equi-width Histogram

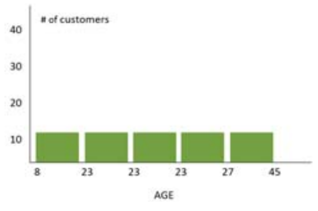

Figure 5: Equi-height Histogram

4.4 Building Equi-height Histogram Using MapReduce

For processing massive data, we build equi-height histograms by using MapReduce. Stats Processor loads the metadata from Metastore and starts up a MapReduce job for each numeric column.

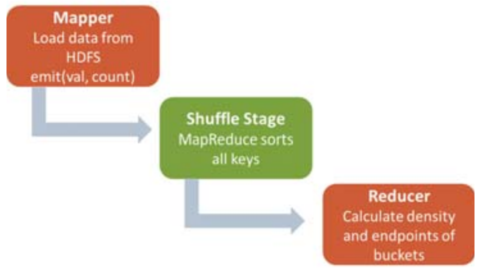

Figure 6: Building Histogram using MapReduce

As shown in Figure 6, the mapper in MapReduce program loads tuples from HDFS and extracts the values from columns. We use the attribute value as output key of mapper to leverage MapReduce shuffle mechanism. It guarantees that the reducer can retrieve keys in order. Therefore, in the reducer, we calculate attribute density value and endpoint of each bucket. Finally, the output of reducer will be stored into Metastore. The endpoints of the histogram in Figure 5 are 8, 23, 23, 23, 27 and 45. The statistics stored in Metastore would help estimate selectivity.

5 Memory Consumption Estimation

5.1 Impala Query Execution

To estimate memory usage of Impala, we need to know how Impala executes our query. In this section, we describe the architecture of Impala and the memory consumption estimation from Impala query execution plan.

5.1.1 The Architecture of Impala

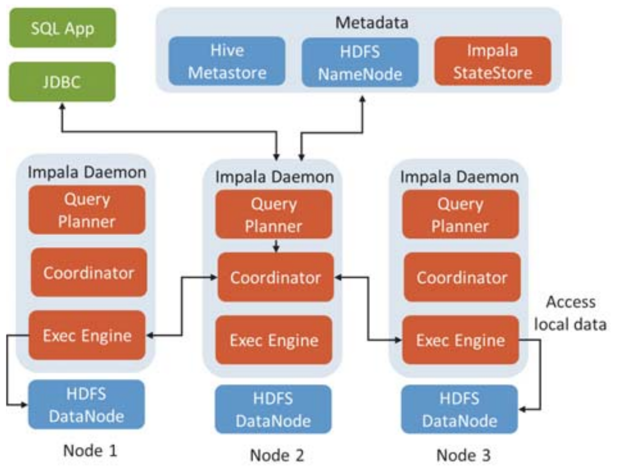

Figure 7: The Architecture of Impala

Figure 7 shows overview of Impala architecture. Impala does not rely on MapReduce to achieve short response time requirement. It directly accesses data on HDFS with a specialized distributed query engine. There are two components in Impala: one is StateStore in the cluster, and the other is Daemon on each HDFS DataNode. StateStore takes responsibility for monitoring the health of each Daemon in the cluster. A SQL application connects to one of Daemons. This Daemon will be the coordinator to generate the query execution plan and coordinate with other Daemons. The exec engines in other Daemons stream the partial results back to the coordinator and the coordinator will response to the user application.

5.1.2 Operations

Select

A select query extracts and filters some data from a table. Impala Daemon scans blocks in HDFS locally and then streams to another node for following operations. Each scan node holds some memory as a buffer. The size of buffer is related to number of blocks in HDFS.

Aggregate Function

The aggregate function like SUM or MIN is performed on separated nodes and then merges the result on an aggregate node.

Join

In general, join operation consumes most of the memory. We focus on the join operation to determine which query can be executed by Impala. Impala uses hash join algorithm to implement the join operation.

As shown in Figure 8, when joining two sets R and S, there are two phases to perform the operation. First, query planner chooses a smaller set R as the input for build phase.

It scans set R and builds the hash table of join attribute. Second, after the hash table is constructed, it scans the larger set S and matches set R by looking up the hash table. The set R must fit into the memory.

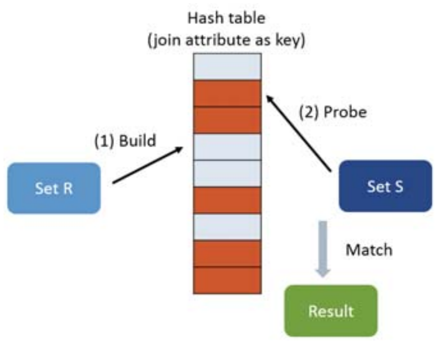

Figure 8: Hash Join

5.2 Estimation from Execution Plan

To estimate the memory consumption for a query, we parse the query execution plan from Impala. Because join operation consumes most of the memory, we focus on it when estimating memory consumption. The hash table is a vector that contains pointers to the table entry data. When the vector is full, it will increase the size automatically. We take the hash table size into account. We set LOAD_FACTOR to 0.75; it means 75% buckets in the hash table contain an entry at least. Each bucket in the vector is a pointer which is 8 bytes in the x64 OS. The estimated memory usage of hash table is calculated in equation (4) and equation (5).

$$\mathrm{mem}_{\mathrm{hashTable}} = 8 * \mathrm{cardinality} * (1/\mathrm{LOAD\_FACTOR}) \quad (4)$$

$$\mathrm{mem}_{\mathrm{join}} = \mathrm{cardinality} * \mathrm{tuple}_{\mathrm{size}} + \mathrm{mem}_{\mathrm{hashTable}} \quad (5)$$

We assume all entries would be distributed to all nodes uniformly, so $\mathrm{mem}_{\mathrm{join}}$ will be equally divided by the number of nodes that involved in processing the join.

As we mention above, we get the buffer size consumed by HDFS scan node from the query execution plan. There are several plan fragments for a query execution plan. Some of them can be executed in parallel, so we add the estimated memory of those plan fragments to calculate the total estimated memory of a node. The value is used to compare with the memory limit of Impala to determine whether the query can be executed by Impala or not.

6 Experiments

In this section, we evaluate the fusion SQL engine system and compare it with Impala and Hive.

6.1 Experiments Settings

The dataset for our experiments is from AWS [3]. The dataset is a database for the bookstore, which consists of three tables: books, customers and transactions. We generate the dataset as shown in TABLE 1. We build equi-height histograms for the dataset.

We run our experiments on a cluster of 5 nodes. One of these nodes functions as the master node, while others nodes are set up as slaves. Because Impala is only supported under the CDH released from Cloudera, we use CDH 5.0.3 as Hadoop environment. The hardware and software configurations are described in TABLE 2 and TABLE 3.

TABLE 1: Dataset Information

Table	Size	#Records
books	8GB	126,227,290
customers	8GB	105,800,433
transactions	16GB	339,799,026

TABLE 2: Hardware Configuration

Device	Description
CPU	Intel Xeon E5520 @ 2.27GHz x 2 with HT
RAM	24GB
Disk	246GB
Ethernet	1Gbps

TABLE 3: Software Configuration

Name	Version
OS	Ubuntu 12.04.4 LTS x64
Apache Hadoop	2.3.0
Apache Hive	0.12.0
Cloudera Impala	1.3.1
MySQL	5.5.37

6.2 Selectivity Estimation

There are 3 queries in TABLE 4. Q1 and Q2 are scan queries with single selection predicate. Q3 is a join query with a selection predicate. We adjust the constant X to show the variation in estimations.

TABLE 4: Queries for Selectivity Estimation

#	Query
Q1	SELECT count(*) FROM books WHERE price < X
Q2	SELECT count(*) FROM transactions WHERE quantity >= X
Q3	SELECT count(*) FROM transactions JOIN books ON (transactions.book_id = books.id AND books.price < X)

From the results shown in Figure 9, Figure 10, and Figure 11, we can find that the estimation values are close to real case values. We also calculate the mean absolute percentage errors (MAPE) in TABLE 5. All MAPE values are under 1.6%.

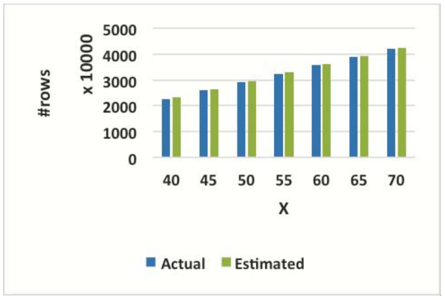

Figure 9: Selectivity Estimation for Q1

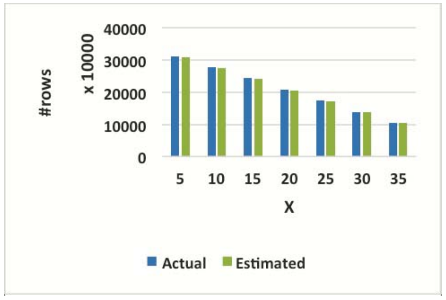

Figure 10: Selectivity Estimation for Q2

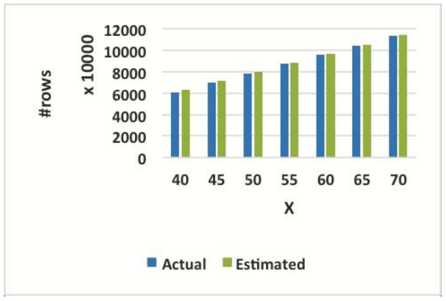

Figure 11: Selectivity Estimation for Q3

TABLE 5: MAPE for Selectivity Estimation

Query	Mean Absolute Percentage Error
Q1	1.565298873
Q2	0.872011458
Q3	1.568515777

6.3 Memory Estimation

In order to evaluate the memory consumption, we implement a monitoring program to monitor each Impala Daemon. When the program starts, it will connect the Impala StateStore to fetch the list of daemons. After the query is executed, monitor starts polling every Impala Daemon per second, and maintains the maximum of memory usage across all nodes. When the query is completed, the monitor will report the maximum memory usage.

To prevent from waiting for a large result set streaming back, we use a limit clause in the SQL. For example, in TABLE 6, we use "LIMIT 10" to fetch the first 10 rows of result.

As illustrated in Figure 12, if the scanning range is small, the data may not occupy all of the buffers. It makes our estimation in those cases slightly higher than actual memory usage. From this experiment result, the MAPE we get is 5.766993529%.

TABLE 6: Query for Memory Estimation

#	Query
Q4	SELECT * FROM transactions JOIN books ON (transactions.book_id = books.id AND books.price < X) LIMIT 10

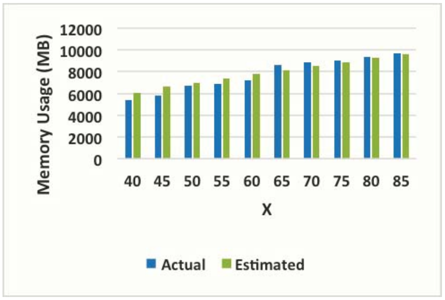

Figure 12: Memory Estimation

6.4 Performance Evaluation

We evaluate the performance of fusion SQL engine system and compare it to that only with Hive or with Impala. We set the memory limitation of Impala and maximum heap size of Hive to 8GB. Hive, Impala and the fusion SQL engine (FSE) will execute all the SQL queries in TABLE 7 for comparison.

Q5 is a normal scanning and aggregation query; Q6 is similar to Q5 with an additional filter predicate; Q7 is a sorting and aggregation query; Q8 causes two large tables join that does not fit in memory. Q9 is a complex query with sub query, sorting, join, aggregation operations.

In Figure 13, for most of the cases (Q5, Q6, Q7, and Q9), the query execution time in FSE and Impala is quite shorter than Hive. However, for the case of Q8, Impala cannot execute it because of the memory size limitation. It will fail at 34.437 seconds. In FSE, it chooses the Hive as the SQL engine to execute the query. Therefore, the performances of our proposed FSE and Hive are similar. The overhead of FSE is under 1 second; it is caused by the simple computation and the communication between Client and Selector.

The speedup is described in TABLE 8. In aggregation operations (Q5 and Q6), we get about 2.53x-5.18x better than Hive, and we get about 3.82x in the join operations (Q9). It shows that the fusion SQL engine system can improve the performance in most of the queries. Our fusion SQL engine can correctly select the best-fit SQL engine (Hive or Impala) to execute a query for a given dataset.

TABLE 7: Queries for Performance Evaluation

#	Query
Q5	SELECT COUNT(*) FROM transactions
Q6	SELECT COUNT(*) FROM customers WHERE name = 'Harrison SMITH'
Q7	SELECT category, count(*) cnt FROM books GROUP BY category ORDER BY cnt DESC LIMIT 10
Q8	SELECT * FROM transactions JOIN books ON (transactions.book_id = books.id AND books.price < 80) LIMIT 10
Q9	SELECT tmp.book_category, ROUND(tmp.revenue, 2) AS revenue FROM (SELECT books.category AS book_category, SUM(books.price * transactions.quantity) AS revenue FROM books JOIN transactions ON (transactions.book_id = books.id AND books.price < 80) GROUP BY books.category) tmp ORDER BY revenue DESC LIMIT 10

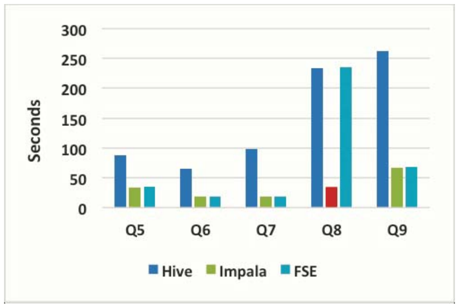

Figure 13: Performance Evaluation

TABLE 8: Speedup of FSE Based on Hive and Impala

#	FSE vs. Hive	FSE vs. Impala
Q5	2.53	0.98
Q6	3.50	0.96
Q7	5.18	0.94
Q8	1.00	N/A
Q9	3.82	0.98

7 Conclusions and Future Work

In this paper, we propose a fusion SQL engine (FSE) system on Hadoop framework. We can correctly select the best-fit SQL engine (Impala or Hive) to execute a query for a given dataset and therefore get the best performance. We also implement a failover mechanism. In the future, we will try to support partition table, add sampling based estimation.

Acknowledgement

The work of this paper is partially supported by Ministry of Science and Technology under contract MOST 103-2221-E-007 -063 -.

Reference

[1] *Apache Hadoop*. http://hadoop.apache.org/

[2] *Impala*. http://impala.io/

[3] *Impala Performance Testing and Query Optimization - Amazon Elastic MapReduce*. http://docs.aws.amazon.com/ElasticMapReduce/latest/DeveloperGuide/impala-optimization.html

[4] *Introducing Cost Based Optimizer to Apache Hive*. Available: https://cwiki.apache.org/confluence/download/attachments/27362075/CBO-2.pdf

[5] A. Abouzeid, K. Bajda-Pawlikowski, D. Abadi, A. Silberschatz, and A. Rasin, "HadoopDB: an architectural hybrid of MapReduce and DBMS technologies for analytical workloads," *Proceedings of the VLDB Endowment*, vol. 2, pp. 922-933, 2009.

[6] M. M. Astrahan, M. W. Blasgen, D. D. Chamberlin, K. P. Eswaran, J. Gray, P. P. Griffiths, *et al.*, "System R: relational approach to database management," *ACM Transactions on Database Systems (TODS)*, vol. 1, pp. 97-137, 1976.

[7] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, pp. 107-113, 2008.

[8] A. Ganapathi, Y. Chen, A. Fox, R. Katz, and D. Patterson, "Statistics-driven workload modeling for the cloud," in *Data Engineering Workshops (ICDEW), 2010 IEEE 26th International Conference on*, 2010, pp. 87-92.

[9] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," in *ACM SIGOPS Operating Systems Review*, 2003, pp. 29-43.

[10] A. Gruenheid, E. Omiecinski, and L. Mark, "Query optimization using column statistics in hive," in *Proceedings of the 15th Symposium on International Database Engineering & Applications*, 2011, pp. 97-105.

[11] A. Lupher, "Shark: SQL and Analytics with Cost-Based Query Optimization on Coarse-Grained Distributed Memory," 2014.

[12] S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, *et al.*, "Dremel: interactive analysis of web-scale datasets," *Proceedings of the VLDB Endowment*, vol. 3, pp. 330-339, 2010.

[13] A. Okcan and M. Riedewald, "Processing theta-joins using MapReduce," in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, 2011, pp. 949-960.

[14] G. Piatetsky-Shapiro and C. Connell, "Accurate estimation of the number of tuples satisfying a condition," in *ACM SIGMOD Record*, 1984, pp. 256-276.

[15] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price, "Access path selection in a relational database management system," in *Proceedings of the 1979 ACM SIGMOD international conference on Management of data*, 1979, pp. 23-34.

[16] Y.-J. Shi, X.-F. Meng, F. Wang, and Y.-T. Gan, "HEDC++: An Extended Histogram Estimator for Data in the Cloud," *Journal of Computer Science and Technology*, vol. 28, pp. 973-988, 2013.

[17] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, 2010, pp. 1-10.

[18] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang, *et al.*, "Hive-a petabyte scale data warehouse using hadoop," in *Data Engineering (ICDE), 2010 IEEE 26th International Conference on*, 2010, pp. 996-1005.

[19] R. S. Xin, J. Rosen, M. Zaharia, M. J. Franklin, S. Shenker, and I. Stoica, "Shark: SQL and rich analytics at scale," in *Proceedings of the 2013 international conference on Management of data*, 2013, pp. 13-24.

[20] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: cluster computing with working sets," in *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, 2010, pp. 10-10.