QMapper: A Tool for SQL Optimization on Hive Using Query Rewriting

Yingzhong Xu[1,2,3]
xuyingzhong@ict.ac.cn

Songlin Hu[2,3]
husonglin@ict.ac.cn

[1]University of Chinese Academy of Sciences, China
[2]Institute of Computing Technology, Chinese Academy of Sciences, China
[3]State Key Laboratory of Software Engineering, Wuhan University, China

ABSTRACT

Although HiveQL offers similar features with SQL, it is still difficult to map complex SQL queries into HiveQL and manual translation often leads to poor performance. A tool named QMapper is developed to address this problem by utilizing query rewriting rules and cost-based MapReduce flow evaluation on the basis of column statistics. Evaluation demonstrates that while assuring the correctness, QMapper improves the performance up to 42% in terms of execution time.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*Parallel databases, Query processing*

Keywords

SQL; Hive; MapReduce; Query Rewriting

1. INTRODUCTION

MapReduce provides a highly simplified programming model, allowing users to run their programs distributedly by implementing Mapper and Reducer functions without caring about the data placement and task scheduling. Hive was in turn developed to provide HiveQL as a high-level language interface and automatically translate it into MapReduce workflows. Both of them are very popular in emerging applications and have been widely used in Internet companies to deal with big data problem [3]. However, when engineers seek to utilize Hive to accelerate analysis applications that previously implemented in RDBMS, they will face the difficulties of migrating current SQL to HiveQL, which is time consuming and neither the correctness nor the performance of the results could be easily guaranteed.

Automatic translation and optimization of SQL to MapReduce mapping attracts lots of attentions from both industry and academia. Current works either adopt a rule-based approach to guide the mapping procedure as done by Google Tenzing or optimized the translation by merging MapReduce jobs to reduce the number of jobs [3]. Both of them focus on optimization at MapReduce level, while ignoring the varieties of the SQL query and their influences on query performance. In this paper, we design a query-rewrite based tool which applies a series of rewriting rules to SQL/HiveQL to guide SQL-to-HiveQL translation and provides a cost-based plan evaluator to choose the optimized equivalent one.

2. METHODOLOGY

We divide the process of QMapper into three phases. For a given SQL query, it is firstly parsed by SQL Interpreter into Query Graph Model (QGM). Then, Query Rewriter applies two series of rules (S-H and H-H) to the QGM to generate equivalent queries. S-H rules are applied to transform the unsupported operators such as (NOT) EXISTS to acceptable form of Hive. Followed by H-H rules that are adopted to generate equivalent HiveQL queries, for instance, exchanging join orders and marking the joining on small tables as map join. Finally, these HiveQL queries generated from the adjusted QGMs are compiled into MapReduce DAGs (direct Direct acyclic graph consists of a set of MapReduce jobs) by Query Compiler. A Plan Evaluator is utilized for estimating these DAGs, so as to identify the best equivalent one.

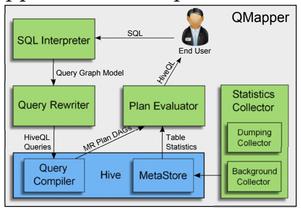

Figure 1: Architecture

2.1 Query Rewriting

Query rewriting aims at generating more equivalent candidate HiveQL queries and increasing the probability of triggering more optimization mapping rules. The more candidates are available, the more chances to find the optimal solution. In QMapper, many well-studied rules [1] were leveraged for translation, and the rewriting rule itself is pluggable to make it easy to extend. Moreover, since query rewriting is performed outside Hive, it can also work together with MapReduce level optimizers.

Consider the following example which retrieves all *Shipments* information of the *Parts* stored in Rio or provided by *Suppliers* in the same city.

```
SELECT * FROM Shipments SP WHERE EXISTS ( SELECT 1
FROM Suppliers S WHERE S.SNO=SP.SNO AND S.city =
'Rio') OR EXISTS ( SELECT 1 FROM Parts P Where
SP.PNO=P.PNO AND P.city='Rio')
```

QMapper will generate a variety of versions of HiveQL that can yield the same results. One version is to divide the two disjunctive predicates into separate SEMI JOIN and then perform an UNION ALL. Another version is to transform the EXISTS to LEFT OUTER JOIN and replace itself with *joinkey* IS NOT NULL. The first one uses SEMI JOIN so that it can remove unnecessary tuples earlier than OUTER JOIN. However, a tuple may satisfy two predicates at same time, and this might bring about duplicates after UNION ALL. Thus, an additional operation is added by QMapper to eliminate the duplicates.

These two execution plans vary a lot and lead to different performance, but it is difficult to intuitively judge which version is cheaper. In order to find the best choice, cost estimation and automatic selection is supported in QMapper.

2.2 Cost Estimation

It is known that the cost formula used in centralized database is not suitable for Hive[2]. Particularly, the intermediate data between Mapper and Reducer or among individual jobs is materialized to disk and transferred via network, and should be considered as a key aspect of cost in IO intensive Hive. Besides, Hiveąŕs optimizer will seek for the chances to merge jobs, e.g. joining 3 tables within a single job when they join on the same key, which also need new cost model. QMapper estimator is thus implemented to estimate the cost of MapReduce DAGs.

A MapReduce DAG consists of a set of MapReduce jobs as nodes, the directed edge between jobs indicates the data flow. A job is composed of Map and Reduce. Each Map and Reduce is considered as a tuple. $MR_i^{M|R} = \{I_i^{M|R}, P_i^{M|R}, O_i^{M|R}\}$. $I_i^{M|R}$ represents the input data of $MR_i^{M|R}$, and $O_i^{M|R}$ represents the output data of $MR_i^{M|R}$. $P_i^{M|R}$ is also a DAG representing the internal operators of Map and Reduce. In order to calculate the cost, we define a profile for each $MR_i^{M|R}$ and $P_i^{M|R}$ to describe the data set being processed. The input or output data profile of an operation is noted as $Profile_{I|O_i}^{M|R|P} = \{ct_{I|O_i}^{M|R|P}, Profile_{I|O_{i_{col(x)}}}^{M|R|P}\}$ where $ct_{I|O_i}^{M|R|P}$ is the number of tuples and $Profile_{I|O_{i_{col(x)}}}^{M|R|P}$ represents the profile of columns in the data. Besides the column-level metrics mentioned in [2], we add ab(Average bytes) for estimating data volumes and quartile value for predicating selectivity. The $Profile_{I_i}^{M}$ of the Maps that use hive tables as input is initialized with table statistics which is collected periodically at background or while loading data. In order to propagate $Profile_{O_i}^{M|R|P}$ which is the profile of output data generated by map, reduce or internal operations based on $Profile_{I_i}^{M}$, we extend the work in [4] to apply profile estimation to the operators within Map and Reduce.

$$Cost(MR) = \sum_{i=1}^{m}(ct_{I_i}^{M} \times \sum_{x=1}^{y} ab_{I_{i_{col(x)}}}^{M} + ct_{I_i}^{R} \times \sum_{x=1}^{y} ab_{I_{i_{col(x)}}}^{R})$$

After the calculation of all profiles of Map and Reduce, QMapper measures the amount of intermediate data generated between Maps and Reduces as well as that produced among different jobs. The best solution with minimize cost is finally selected.

3. EXPERIMENTS

We evaluated our system on a cluster consisting of 13 nodes, each of them has 8 cores and 16GB RAM. All nodes use CentOS 6.2, Java 1.6.0_22 and hadoop1.0.1. The latest Hive 0.10.0 was deployed and 15GB TPC-H data set was generated as workload. We choose relatively complex queries: Q2, Q18 and Q10 from TPC-H, which provide opportunities to generate more equivalent queries to verify the effectiveness of QMapper.

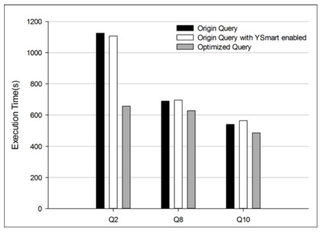

Figure 2: Execution time

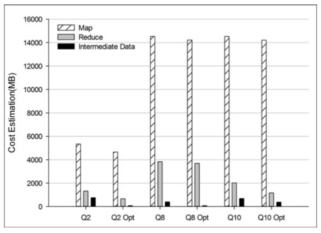

Figure 3: Cost estimation

We compared the execution time of 3 SQL queries translated manually in the most direct way with that translated by our tool and YSmart respectively. Figure 2 shows that an average performance improvement of 20.2% is gained by QMapper, while YSmart is not able to optimize these queries. Figure 3 indicates the total cost estimation of optimized queries and the original ones. As shown in Figure 3, the intermediate data generated between each job is reduced by 89.7% for query Q2, and that is why 41.7% of execution time is saved. For Q8 and Q10, their performance are improved by 8.9% and 10.2% respectively, in that their structures have been already close to the optimized queries and the input tables are too big compared to the intermediate data set, thus a saving on intermediate data does not help much.

4. CONCLUSION AND FUTURE WORK

A tool for translating SQL queries to optimized HiveQL is introduced together with its two key components: Query Rewriter and Plan Evaluator. We are extending the Rewriter to support more rules and enhancing the estimator to involve the parallelism of multiple SQL blocks, data compression and the difference of cost effect between Map and Reduce to provide a more fine-grained cost model. SQL Optimization of HiveQL compatible system, like Shark, is also our future work.

5. ACKNOWLEDGMENTS

For the completion of this research, we would like to thank the National Natural Science Foundation of China under Grant No.61070027, 61020106002, 611611605. This work is also supported by State Key Laboratory of Software Engineering (SKLSE2012-09-02).

6. REFERENCES

[1] U. Dayal. Of nests and trees: A unified approach to processing queries that contain nested subqueries, aggregates, and quantifiers. In *Proc. VLDB*. Morgan Kaufmann Publishers Inc., 1987.

[2] A. Gruenheid, E. Omiecinski, and L. Mark. Query optimization using column statistics in hive. In *Proc. IDEAS*. ACM, 2011.

[3] R. Lee, T. Luo, Y. Huai, F. Wang, Y. He, and X. Zhang. Ysmart: Yet another sql-to-mapreduce translator. In *Proc. ICDCS*, pages 25–36. IEEE, 2011.

[4] M. V. Mannino, P. Chu, and T. Sager. Statistical profile estimation in database systems. *ACM Comput. Surv.*, 20(3):191–221, Sept. 1988.