International Journal of Grid Distribution Computing
Vol. 8, No.4, (2015), pp. 249-256
http://dx.doi.org/10.14257/ijgdc.2015.8.4.24

# An Optimized Method of Translating SQL to More Efficient Map-reduce Tasks

Jin Cao[1], Honglin Han[2], Mingming Zhao[1], Sijing Ye[1], Dehai Zhu[1] and Lin Li*

[1] *College of Information and Electrical Engineering, China Agricultural University, Beijing, China*
[2] *Beijing GeoBeans Information Technology Co. Ltd*
*Email：lilincau@126.com*

***Abstract***

*Today it is common for map-reduce programs to be created from higher-level programming systems, which is often an implementation of SQL. In this paper, we propose an optimized method of translating SQL query to map-reduce tasks, which can be performed more efficiently by distributed computing systems. Our method introduces an optimization strategy which is generally applied in the procedure of merging query-plan-tree, an intermediate data structure derived from SQL abstract-syntax-tree, and this mainly helps us to achieve a considerable reduction on the number of generated map-reduce tasks and efficiency improvement of map-reduce implementation. Experimental evaluations performed by means of TPC-H tests show that our method significantly outperforms Hive in terms of reliability and efficiency.*

***Keywords:*** *SQL, map-reduce, query-plan-tree, optimization*

## 1. Introduction

We are currently in the era of "Big Data". Organizations as well as individuals continuously generate and collect enormous amounts of data covering almost all aspects of life and work. Facebook adds 15 terabytes of data each day into its petabyte-scale data warehouse [1]. Powerful telescopes in astronomy, particle accelerators in physics, and genome sequencers in biology are placing massive volumes of data into the hands of scientists. There is a growing need for ad hoc analysis of extremely large data sets, especially at Internet companies where innovation critically depends on the ability to analyze terabytes of data collected every day [2]. The ability to perform scalable and timely analytic processing of these datasets to extract useful information is now critical for success [3]. Map-reduce is a style of computing that has been implemented in several systems, including Google's internal implementation (simply called map-reduce) and its widespread open-source implementation Hadoop [4]. The map-reduce framework splits the input dataset into smaller partitions, and spawns multiple copies of maps and reduces to operate on each partition in parallel [7]. Implementations of map-reduce enable many of the most common calculations on large-scale data to be performed on computing clusters efficiently and in a way that is tolerant of hardware failures during the computation. Users generally only need to write two functions, namely map and reduce, while the system manages the parallel execution, coordination of tasks which execute map or reduce, and deals with the possibility of task failure [4]. However, when it comes to the specific implementation of map-reduce applications, there should still be a considerable amount of programming work, which may generally be carried out in a manual way.

Today, it is a very prevalent practice for various applications to create map-reduce programs from still higher-level programming systems, often an implementation of SQL. The map-reduce paradigm is too low-level and rigid, and leads to a great deal of custom

ISSN: 2005-4262 IJGDC
Copyright ⓒ 2015 SERSC

user code which is difficult to maintain and reuse [2]. Consequently, to ensure that the use of map-reduce is more convenient and available, many enterprises and institutions have dedicated themselves to providing higher-level programming interfaces for map-reduce. Olston et al. [2] on Yahoo! Research designed a new language called Pig Latin, to fill in the gap between the declarative style of SQL, and the low-level, procedural style of map-reduce. The accompanying system, Pig, is fully implemented, and compiles Pig Latin programs into (ensembles of) map-reduce jobs which are executed using Hadoop. Hive, another open-source data warehousing solution built on top of Hadoop and designed by Thusoo et al. [1] on the Facebook Data Infrastructure Team, supports queries expressed in an SQL-like declarative language called HiveQL, which is compiled into map-reduce jobs executed on Hadoop. In addition, HiveQL supports custom map-reduce scripts to be plugged into queries. Tenzing is an SQL query execution engine built on top of MapReduce for ad hoc analysis of Google data, and is designed by Chattopadhyay et al. [5]. Tenzing, which supports a mostly complete SQL implementation (with several extensions), has been widely adopted inside Google, and serves 10000+ queries per day over 1.5 petabytes of compressed data in several different data sources and formats.

Many attempts have been made to create simpler interfaces on top of map-reduce, but, to the best of our knowledge, such implementations either suffer from low efficiency and poor SQL compatibility, such as Pig and Hive, or being only available for internal use, such as Tenzing. Part of their inefficiency is the product of map-reduce's perceived inability to exploit relevance among various operations inside a single SQL query. It is quite pervasive to execute operations more than once on the same table in a SQL query, such as the implementation of self-intersection, and each of these operations requires a corresponding map-reduce task to accomplish this. A map-reduce task generally stores intermediate data and work file locally, and such activities are constantly causing network transmission as well as I/O overhead. All the factors mentioned above consequently contribute to the inefficiency of the entire system. Given the fact that most of the existing solutions adopt this one-operation-to-one-task translation mode, in this paper, we propose an optimized method of translating SQL query to map-reduce programs, which takes full advantage of the relevance among various operations within a single SQL query. Our method introduces an optimizing strategy which is generally applied in the procedure of merging query-plan-tree, an intermediate data structure derived from SQL abstract-syntax-tree. This mainly contributes to the achievement of a considerable reduction in the number of generated map-reduce tasks and efficiency improvement of map-reduce implementation. In the evaluations, our results significantly outperform Hive, due to the fact that. Although Hive is optimal in terms of language abstraction, it is not optimal with respect to how the system makes use of the internal relevance of each query.

The remaining part of this paper is organized as follows. Section 2 describes the details of our method, especially the optimizing strategy by utilizing the SQL query-plan-tree. Section 4 presents the comparative experimental results between our method and Hive by performing TPC-H tests, as well as accompanying empirical analysis. Section 5 concludes the paper.

## 2. Translating SQL to Map-Reduce

Most of the existing solutions adopt the one-operation-to-one-task translation method when creating map-reduce programs in the implementation of SQL, and generally attach little importance to the relevance among various operations within a single SQL query. Since each map-reduce task stores intermediate data and work file on a local disk, such activities constantly cause network transmission as well as I/O overhead. In consideration of the circumstances mentioned above, our research focuses on achieving the efficiency improvement of map-reduce implementation by reducing the quantity of map-reduce tasks generated. We apply an optimizing

 Copyright ⓒ 2015 SERSC

strategy in the procedure of merging query-plan-tree, which contributes to the achievement of a considerable reduction in the number of generated map-reduce programs.

### 2.1. Query-Plan-Tree

We use Antlr to accomplish the foundation work, including lexical analysis and syntactic analysis of each SQL query. Then we determine its abstract-syntax-tree (referred to simply as AST in the remaining parts of this paper) as an outcome. However, AST, which does not provide a comprehensive expression of the data flow within each query, cannot be applied directly to the generation of map-reduce programs. Therefore, we define the query-plan-tree as a new type of intermediary data structure derived from AST, which can generally provide a comprehensive representation of data flow within each query. Due to the fact that a typical SQL query may generally contain operations such as selecting, projecting, aggregating, and jointing and sorting, we define five types of operation node, including *SelectNode*, *WhereNode*, *OrderbyNode*, *GroupbyNode* and *TableNode*, to perform the generalization of each query, and thereby facilitating the induction of its AST.

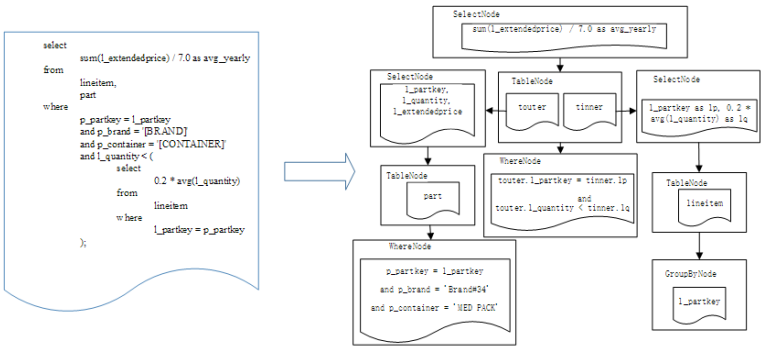



**Figure 1. Abstracted Structure of TPC-H Query no. 17 Abstract-Syntax-Tree, Translation from Query to AST**

*SelectNode* stores the fields that we intend to acquire through the query, in which it may be either individual or an outcome of aggregation function. *WhereNode* stores fields information of filtrating action prescribed by the WHERE clause. *OrderbyNode* stores fields information regarding sorting action, such as sort direction and related tables. *GroupbyNode* stores fields information of aggregation action. *TableNode* stores the information of all the tables, which may be either individual or multiple, as well as temporary tables generated by selecting operation. In this manner, the structure of AST can be converted into an abstracted format, which consists of operation nodes and contains the key components of each query, and in this situation *TableNode* enables us to handle nest queries effectively. We use the example of *TPC-H* query no. 17 to illustrate the method of parsing SQL query to AST, and the process is illustrated in Figure 1.

After the accomplishment of the query parse, the abstracted query AST must still be inducted further into the query-plan-tree, which can be utilized to generate map-reduce programs. Each query can be generally deconstructed into five types of operations, namely selecting, projecting, aggregating, and jointing and sorting, thus we define three types of tasks to execute the map-reduce implementation. The AGG task is intended to perform aggregating and grouping operations, while the Join task is designed to perform the jointing operation of the incoming data. The Sort task is used to perform data sorting. We then use AGG, Join and Sort as the three types of task nodes to establish the query-plan-tree.

Copyright ⓒ 2015 SERSC

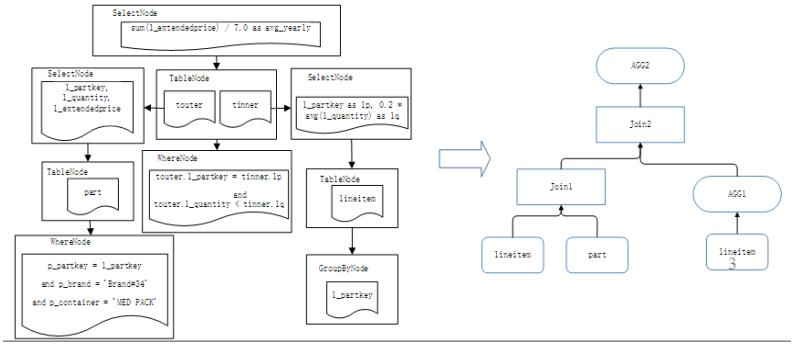



**Figure 2. Abstracted Structure of TPC-H Query no. 17 Query-Plan-Tree, Transformed from AST.**

The process of transforming TPC-H query no. 17 from its AST to query-plan-tree is shown in Figure 2.

### 2.2. Optimization Strategy

Due to the fact that it is quite pervasive to execute operations more than once on the same table in an SQL query, such as the execution of self-intersection, with a full awareness that the inability to exploit relevance among various operations inside a single query constantly affects the efficiency of the existing systems, we propose an optimization strategy which takes full advantage of these factors, and is aimed at achieving efficiency improvement of map-reduce implementation. The map-reduce program should be generated for each of the task nodes in the query-plan-tree. Thus the simplification of the query-plan-tree would certainly conduce to the reduction of the number of map-reduce programs. In addition, the operation relevance within the query should be helpful to the simplification of the query-plan-tree. Our research focuses on two cases of operational relevance:

*Case 1.* Incoming data of multiple task nodes originate from exactly the same table, and their corresponding map-reduce program applies the identical key to perform partition in the procedure of mapping.

*Case 2.* The parent task node and child task node share the same key to perform partition in the procedure of mapping.

Then we utilize these two relevance to accomplish the simplification of the query-plan-tree. We design our optimizing strategy as follows: when the relevance mentioned above is detected, there are several generalized rules to be complied with so that we may merge multiple task nodes into a single one.

*Rule 1.* If two arbitrary task nodes are intended to acquire incoming data from the same table, and their corresponding map-reduce program applies the same key to perform partition in the procedure of mapping, then these two task nodes should be incorporated into a single one. This newly incorporated task node should perform the pre-processing of the incoming data, which was required by the previous two original task nodes in the procedure of map, and process the intermediate data according to the aforementioned two original task nodes in the procedure of reducing.

*Rule 2.* If an arbitrary task node and its child task nodes share the same key to performing partition in the procedure of mapping, we should first merge its child nodes according to Rule 1, and only if this task node has multiple child nodes may we assign the processing of the parent node to the newly incorporated task node derived from the previous child nodes.

 Copyright ⓒ 2015 SERSC

As showed in the process of simplification of TPC-H query no. 17 query-plan-tree in Figure 3, the quantity of the task nodes was reduced significantly from 4 to 2.

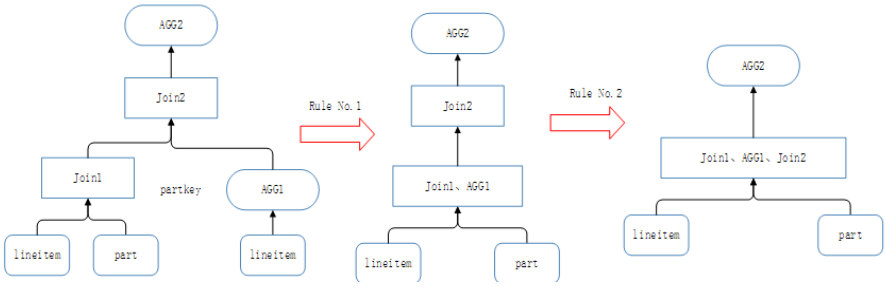



**Figure 3. Simplification Process of TPC-H Query no. 17 Query-Plan-Trees**

Above all, we take advantage of the relevance within the query and apply the optimization strategy to the simplification of the query-plan-tree, thus the generated map-reduce workflow generally performs no more repeat reading of the same table for data acquisition. In addition, by performing task node merging, the network transmission and I/O overhead decrease significantly along with a considerable reduction on the quantity of map-reduce tasks, and all these consequently contribute to the efficiency improvement of the entire system.

## 3. Experimental Evaluation

*TPC-H* is a decision support benchmark which consists of a suite of business oriented ad hoc queries and concurrent data modifications. The queries and data populating the database have been chosen to have broad industry-wide relevance. In this paper, we adopt *TPC-H* as the test case to conduct the comparison experiment between Hive and SQL-MR, in which the new strategy would be fully implemented, so that the high efficiency and reliability of our method could be verified.

### 3.1. Experimental Instruments

As may be seen from the structure shown in Figure 4, the hardware environment of our experiment is a computing cluster consisting of six nodes, with one name node and one job tracker. The other four serve as data nodes and task trackers at the same time. Each of the computing nodes is equipped with a Xeon 3065 CPU, a memory of 2 gigabytes and a hard disk of 360 gigabytes.

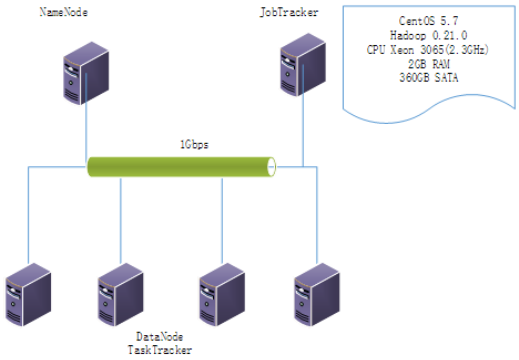



**Figure 4. Configuration of Computing Cluster Applied In Our Environment**

Copyright ⓒ 2015 SERSC

### 3.2. Functional Test

Before the efficiency comparison, we firstly conduct a functional test of SQL-MR on the data set of TPC-H to verify the fundamental function of our system. The test mainly includes basic SQL query operations, which would be carried out in circumstances of single table, multi tables as well as nested tables, such as select, join and aggregate function etc. The test results in table 1 showed that our system is fully functional.

**Table 1. Results of Functional Test on SQL-MR**

| Test Function | Single-table operation | Multi-table operation | Nested operation |
|---|---|---|---|
| SELECT | √ | √ | √ |
| WHERE | √ | √ | √ |
| ORDER BY | √ | √ | √ |
| SUM | √ | √ | √ |
| AVG | √ | √ | √ |
| MIN | √ | √ | √ |
| MAX | √ | √ | √ |
| GROUP BY | √ | √ | √ |
| INNER JOIN | N/A | √ | √ |

### 3.3. Efficiency Evaluation Results

In this experiment, we applied TPC-H queries no. 17 and 18 to perform the comparison test between Hive and SQL-MR and we use the time consumption of each system to perform the query as the efficiency indicator.

**Figure 5. Time Consumption of Each System for performing every Map-reduce Task Generated from TPC-H Queries no. 17 and 18**

We then respectively measure the time consumption that each system used to perform every map-reduce task generated from TPC-H queries no. 17 and 18. It is clear that our method generates fewer map-reduce tasks, and its preformation on individual tasks are also less time-consuming. Both these circumstances indicate that our method performs more efficiently than Hive.

## 4. Conclusion and Future Work

The evaluation results indicate that our method performs more efficiently compared to Hive, a typical instance of the existing systems. We thoroughly studied and took advantage of the relevance among various operations within a single SQL query, to which


Copyright ⓒ 2015 SERSC

existing systems generally attach little importance, thus a considerable reduction on the number of generated map-reduce tasks and the accompanying efficiency improvement of map-reduce implementation may be achieved.

Though a SQL to map-reduce translating system based on our method was implemented and functioned properly, there is still several deficiency in our work. For one thing, some standard SQL function such as LIKE and IN do not get supported by SQL-MR, in other words, this system was partially implemented and required follow-up development. Moreover, our preceding work mainly focused on the optimization strategy which was intended for the reduction of network transmission and reusability of intermediate data. And we have every reason to believe that there should be other constraint factors for the generating and conducting of map-reduce when it comes to the implementation of SQL. Anyway, there is additional optimization effort to be devoted to.

## Acknowledgements

This paper is supported by the "National Key Technology R&D Program under the Twelfth Five-Year plan of P.R. China" (Grant No. 2012BAK19B04-03).

## References

[1] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff and R. Murthy, "Hive, a warehousing solution over a map-reduce framework", Proceedings of the VLDB Endowment, vol. 2, no. 2, **(2009)**.

[2] C. Olston, B. Reed, U. Srivastava, R. Kumar and A. Tomkins, "Pig latin, a not-so-foreign language for data processing", Proceedings of the 2008 ACM SIGMOD international conference on Management of data, **(2008)** June 10-12, Vancouver, Canada.

[3] S. Babu, "Towards automatic optimization of Map Reduce programs", Proceedings of the 1st ACM symposium on Cloud computing, **(2010)** June 10-11, Indianapolis, USA.

[4] A. Rajaraman and J. D. Ullman, "Mining of massive datasets", Cambridge University Press, Cambridge **(2012)**.

[5] L. Lin, V. Lychagina, W. Liu, S. Mittal, P. Aragonda, B. Chattopadhyay, Y. Kwon and M. Wong, "Tenzing a sql implementation on the map reduce framework", Proceedings of the VLDB Endowment **(2011)**.

[6] J. Dean and S. Ghemawat, "Map Reduce, simplified data processing on large clusters", Communications of the ACM, vol. 1, **(2008)**, pp. 51.

[7] J. Tan, S. Kavulya, R. Gandhi, S. Kavulya and P. Narasimhan, "Visual, log-based causal tracing for performance debugging of map reduce systems", Distributed Computing Systems (ICDCS), 2010 IEEE 30th International Conference on. **(2010)** June 21-25, Genova, Italy.

[8] M. Stonebraker, D. Abadi, D. J. DeWitt, S. Madden, E. Paulson, A. Pavlo and A. Rasin, "Map Reduce and parallel DBMSs, friends or foes", Communications of the ACM, vol. 1, **(2010)**, pp. 53.

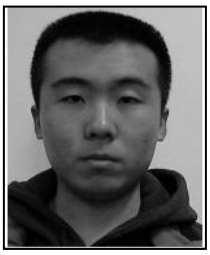
[9] A. F. Gates, O. Natkovich, S. Chopra, P. Kamath, S. M. Narayanamurthy, C. Olston, B. Reed, S, Srinivasan and U. Srivastava, "Building a high-level dataflow system on top of Map-Reduce, the Pig experience", Proceedings of the VLDB Endowmentm, vol. 2, **(2009)**, pp. 2.

[10] M. L. Mchome, "Comparison Study between MapReduce (MR) and Parallel Data Management Systems (DBMs) in Large Scale Data Analysis", MSCS Honor Projects, Macalester University, **(2011)**.

[11] S. Connolly, S. Wooledge and T. Aster, "Harnessing the value of big data analytics", Big Data Analytics, **(2011)**.

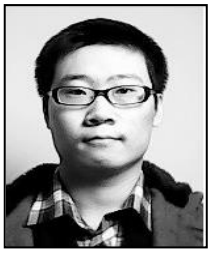
Copyright ⓒ 2015 SERSC 

## Authors



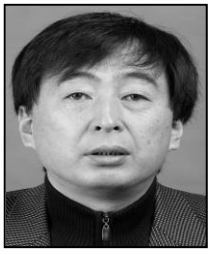
**Jin Cao**, received his Bachelor degree in Computer Science from College of Information and Electrical Engineering, at China Agricultural University, China in 2013. Currently, He is a candidate for the degree of Master of Computer Science in College of Information and Electrical Engineering of China Agricultural University. His current research interest includes High Performance Computing, Remote Sensing Image Processing, and Geographic Information System etc.

**Honglin Han**, works as computer engineer of Beijing GeoBeans InfoTech Co. He obtained his Master Degree from China Agricultural University, China. His current job mainly includes address matching, remote sensing image processing etc..



**Mingming Zhao**, received his Bachelor degree in Information and Computing Science from at Taiyuan University, China in 2013. Currently, he is a candidate for the degree of Master of Computer Science and Technology in China agricultural university of College of Information and Electrical engineering. His research interests include Image Process etc.

**Sijing Ye**, was from Anhui province, China and became a PhD student of College of Information and Electrical Engineering in China Agricultural University in 2012. His current research interest focus on application of spatial information technologies on domain of agricultural informatization



**Dehai Zhu**, works as professor of Geographic Information Science Department, College of Information and Electrical Engineering at China Agricultural University, China. His current research interest includes information management of Land Resources and agricultural informatization.

**Lin li**, works as professor of Computer Science and Technology Department, College of Information and Electrical Engineering at China Agricultural University, China. She obtained her Ph.D. from China Agricultural University, China. Her current research interest includes software engineering (SE), software automation (SA), and mobile internet technology and so on. Until now, she has published more than 50 peer-reviewed papers.

 Copyright ⓒ 2015 SERSC