The Journal of Systems and Software 125 (2017) 35–46

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Logical query optimization for Cloudera Impala system☆

Jiaoyang Ma[a], Ling Chen[a,*], Mingqi Lv[c], Yi Yang[a], Yuliang Zhao[a], Yong Wu[b], Jingchang Wang[b]

[a] College of Computer Science and Technology, Zhejiang University, Hangzhou, Zhejiang, China
[b] Zhejiang Hongcheng Computer Systems Co., Ltd., Hangzhou, Zhejiang, China
[c] College of Computer Science and Technology, Zhejiang University of Technology, Hangzhou, Zhejiang, China

ARTICLE INFO

Article history:
Received 29 July 2015
Revised 14 September 2016
Accepted 23 November 2016
Available online 24 November 2016

Keywords:
Logical query
Bushy tree
Cost model
Cloudera Impala system

ABSTRACT

Cloudera Impala, an analytic database system for Apache Hadoop, has a severe problem with query plan generation: the system can only generate query plans in left-deep tree form, which restricts the ability of parallel execution. In this paper, we present a logical query optimization scheme for Impala system. First, an improved McCHyp (MinCutConservative Hypergraph) logical query plan generation algorithm is proposed for Impala system. It can reduce the plan generation time by introducing a pruning strategy. Second, a new cost model that takes the characteristics of Impala system into account is proposed. Finally, Impala system is extended to support query plans in bushy tree form by integrating the plan generation algorithm. We evaluated our scheme using TPC-DS test suit. Experimental results show that the extended Impala system generally performs better than the original system, and the improved plan generation algorithm has less execution time than McCHyp. In addition, our cost model fits better for Impala system, which supports query plans in bushy tree form.

1. Introduction

As the era of big data is coming, how to query more efficiently becomes an urgent need for industries, e.g., Internet, telecommunication, and finance. To meet this need, analytic database systems for big data appear, e.g., Google Dremel system (Melnik et al., 2010), Berkeley Shark system (Engle et al., 2012), and Cloudera Impala system. Instead of using the MapReduce batch processing (Dean and Ghemawat, 2008), Impala system uses a distributed query engine, which gets data directly from HDFS (Hadoop Distributed File System) (White, 2012) or HBase (George, 2011), in order to reduce the query response time. As compared with the distributed data warehouse Hive (Thusoo et al., 2009; Thusoo et al., 2010), the query response time is reduced by a factor of 3–90. Impala system's beta release was in October 2012 and its GA (Generally Available) edition was in May 2013. The most recent version, Impala system 2.0, was released in October 2014. Impala system's ecosystem momentum continues to accelerate with nearly one million downloads since its GA (Marcel et al., 2015).

However, the current Impala system has the following problems: First, Impala system can only generate query plans in left-deep tree form, which can only be executed serially. A node in the plan tree cannot be executed unless its left sub-tree has been executed (the right sub-tree is always a scan node, which only reads data from storage). Second, Impala system uses a cost-based logical query plan generation algorithm to generate logical query plans. However, the cost model only takes the size of data and the cardinality of joins into account. This cost model is not accurate in cluster environment, since the network transfer cost has not been considered. When Impala system runs on a cluster that has a slow network transfer rate, the network transfer cost can be even higher than disk I/O. Ignoring the network transfer cost may lead to sub-optimal plans.

For the first problem, we exploit query plans in bushy tree form with the bushy tree based logical query optimization, which enables concurrent execution of nodes in a query tree. For the second problem, we propose a new cost model, which considers the characteristics of Impala system. It is more accurate for a plan tree, and can lead to an optimal query plan.

In this paper, we present a logical query optimization scheme for Impala system. First, an improved McCHyp (Fender and Moerkotte, 2013) logical query plan generation algorithm (called Improved-McCHyp) is proposed for Impala system. A pruning

☆ This work was funded by the Ministry of Industry and Information Technology of China (No. 2010ZX01042-002-003-001), China Knowledge Centre for Engineering Sciences and Technology (No. CKCEST-2014-1-5), the National Natural Science Foundation of China (Nos. 60703040, 61202282, and 61332017), the Science and Technology Department of Zhejiang Province (Nos. 2011C13042, 2013C01046, and 2015C33002), the Natural Science Foundation of Zhejiang Province (No. LY15F020025).

* Corresponding author.
E-mail address: lingchen@cs.zju.edu.cn (L. Chen).

http://dx.doi.org/10.1016/j.jss.2016.11.038
0164-1212/

strategy is introduced to reduce plan generation time. Second, a new cost model is proposed, which considers disk I/O, network transfer cost, the size of the right table in the join operation, and the size of join. The cost model is based on bushy tree, which suits for the modified Impala system. Finally, Impala system is extended to support bushy tree query plans and the plan generation algorithm is integrated into Impala system.

Our contributions are as follows.

- Present a query optimization scheme for Impala analytic database system. The scheme considers the characteristics of Impala system and prefers bushy tree logical query plan that makes full use of parallel execution so as to accelerate query processing.
- Present the Improved-McCHyp algorithm, which is optimized by a pruning strategy to reduce logical query plan generation time.
- Design a bushy tree oriented cost model that takes into account the characteristics of Impala system, i.e., disk I/O, network transfer cost, the size of right table, and the size of join.
- Implement the scheme in Impala system and conduct extensive experiments. The results demonstrate the validity and efficiency of our algorithm.

The rest of this paper is organized as follows. Section 2 provides related work, and Section 3 introduces several preliminaries. Section 4 details the logical query optimization scheme, including the Improved-McCHyp algorithm, the cost model, and the bushy tree based logical query plan presentation. Section 5 contains experimental evaluation, and Section 6 concludes the paper.

2. Related work

Logical plan generation can be divided into traditional plan generation and heuristic plan generation. When using traditional plan generation approaches, plans are represented by trees in most cases. The shape of trees can be further divided into two categories: left-deep tree and bushy tree. Steinbrunn et al. (1997) analyzed the search space for left-deep tree and bushy tree, and drew a conclusion that for n relations, there should be $n!$ possible results based on left-deep tree, while $\binom{2(n-1)}{n-1}(n-1)!$ possible results based on bushy tree. To traverse the tree, there are two approaches: top-down traversing through memorization and bottom-up traversing via dynamic programming. A lot of work has been done in these areas.

2.1. Heuristic plan generation

Heuristic plan generation algorithms include genetic algorithm, particle swarm algorithm, ant colony algorithm, etc., which can generate near optimal plans and have acceptable time and space complexity. Viglas and Naughton (2002) proposed a rate-based optimization framework that aims at maximizing the output rate of query evaluation plans. They use two heuristic algorithms, local rate maximization and local time minimization, to find locally better plans. The proposed cost model has two parts, namely the cost of handling an input from the left stream and the cost of handling a right stream input. This rate-based optimization works with infinite input streams, which is different with Impala system. Gruenheid et al. (2011) proposed a permutation algorithm to optimize queries for Hadoop MapReduce framework. Their assumption is that in the framework, additional cost is incurred when tuples have to be written back to disk in the reduce phases and read again from disk in the next map phases. Therefore, the cost-based optimizer sums up the cardinalities of all joins, which generate tuples that have such I/O operations. They also introduce an extension to the Hive Metastore, which stores metadata that have been extracted on the column level of the user database. This kind of extension is helpful for our optimization, as Impala system also uses Hive Metastore as its metadata management model. Ganguly et al. (1996) presented two heuristic cost functions, which aim at achieving efficient cost models that accurately approximate the response time of parallel query executions. They also approximated the optimal degree of the parallelism of each node in the operator tree. Sevinç and Coşar (2010) proposed a GA (Genetic Algorithm) based query optimizer for distributed QEPs (Query Execution Plans) generated by NGA (New Genetic Algorithm) on their departmental cluster machine (Onder, 2010) and came up with accurate communication cost formulas. The cost model takes communication cost and processing cost into account, and the communication cost is the maximum cost among all nodes. Golshanara et al. (2014) proposed a multi-colony ant algorithm for optimizing join queries in cluster environment, where relations can be replicated but not fragmented. In addition, two cost models (one based on total time, and the other based on response time) are used, which considered both communication cost and local processing cost. The algorithm can be improved by using learning automata to adjust the algorithm parameters, which is more promising than finding the parameters by trial and error.

2.2. Left-deep tree plan generation

Before the 1990s, researchers mainly paid attention to left-deep tree based query optimization, because database applications were stand-alone and left-deep tree based query plan had the advantage of small search space and short optimization time. Recently, some applications also prefer left-deep tree to reduce plan generation time when there are many relations in the join.

Cluet and Moerkotte (1995) raised an algorithm that is more efficient than dynamic programming, and showed that the problem of constructing optimal left-deep processing trees for star queries is NP-complete. Zhou et al. (2014) proposed an improved algorithm for DPhyp (Moerkotte et al., 2008) to shrink the search space from bushy tree to left-deep tree, and integrated it into Impala system, which shortens query time by 67%–80%. They also proposed a cost model that took communication cost and tables in the join into account, which was suitable for normal distributed systems. However, due to the parallel execution characteristics of Impala system, the effect of optimization has limitations for left-deep search space, and the cost model does not fit for Impala system when considering the execution flow depicted in Section 3.4.

2.3. Bushy tree plan generation

As distributed systems appeared, researchers paid more attention to bushy tree based query optimization. This is because that the child nodes of the same parent could run parallel at different nodes, so the query efficiency could be improved.

Many top-down join optimization algorithms that generate query plans in bushy tree form have been proposed recently. DeHaan and Tompa (2007) gave a top-down join enumeration algorithm MinCutLazy for connected graphs. They improved query optimization efficiency by integrating branch-and-bound algorithm. This algorithm performs well for acyclic query graphs, but the complexity is high when dealing with cyclic query graphs. Fender and Moerkotte (2012) presented an improved version MinCutLazyImp, which removed unnecessary steps and led to better performance. However, this algorithm is difficult to implement and there are some unnecessary calculations during the join enumeration procedure. Fender et al. (2012) optimized the pruning strategy and proposed a new top-down algorithm MinCutConservative, which is easier to implement and a bit faster than MinCutLazy.