An Analytical SQL Engine for MapReduce

Shengsheng Huang, Jason Dai, Zhihui Li
Intel Corporation

1. Introduction

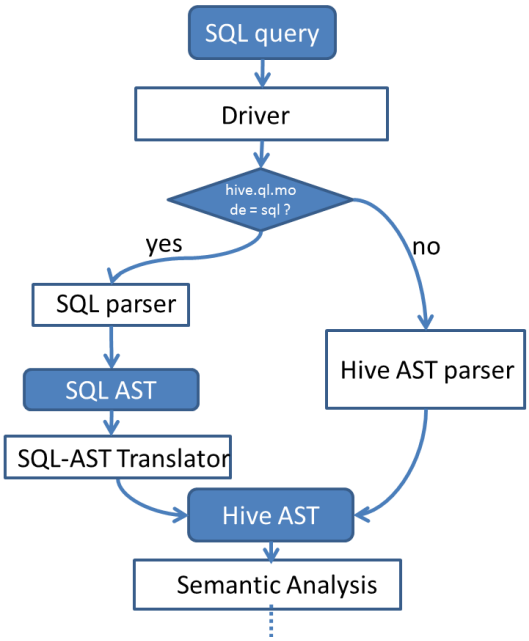
Higher level languages or APIs (such as Pig, Hive and Cascading) significantly lower the barrier to MapReduce based data analytics. On the other hand, SQL is still the most important query language in modern business application environment, with more than 30 years of investments by the industry. There is a wealth of business users, enterprise analytics applications and third-party tools (such as query builders and BI applications) that all require full SQL support.

While there are continuous efforts in extending Hive's SQL support (e.g., see some recent examples such as HIVE-2005 and HIVE-2810), many widely used SQL constructs are still not supported in Hive, such as selecting from multiple tables, subquery in WHERE clauses, etc.

We propose to build a SQL-92 compatible engine (for MapReduce based analytical query processing) as an extension to Hive, which allows the re-use of both the support of the subset of SQL-92 in Hive and HiveQL extensions.

2. High Level Design

At a high level, the SQL engine will use an open source SQL parser (https://github.com/porcelli/plsql-parser) to generate AST (abstract syntax tree) for the SQL query, then transform the SQL AST to MapReduce jobs through a series of context-aware analyses and optimizations (reusing the semantic analyzer and optimizer in Hive when appropriate), as illustrated in the figure below.



The SQL frontend will co-exist with the HiveQL frontend; consequently, one can even mix SQL and

HiveQL statements in their queries (switching between HiveQL mode and SQL-92 mode using a "hive.ql.mode" parameter before each query statement).

2.1 First stage – SQL parser

In the first stage, the SQL string will be tokenized and parsed into an internal AST format – referred as **SQL-AST**, using an open source SQL parser (https://github.com/porcelli/plsql-parser)

2.2 Second stage – SQL-AST translator

In the second stage, the SQL-AST will be analyzed and optimized to generate the output Hive AST. While more details can be found in the related design for the sub-tasks (e.g., subquery in WHERE clauses), a brief overview of this stage is provided below.

The SQL-AST translator will first analyze the SQL-AST and collect information to build necessary contexts. In particular, it needs to build a *QueryInfo* tree and a *FilterBlock* tree for each statement for subquery support. The QueryInfo tree maintains the information of each query/subquery and the nested relationships between subqueries. The FilterBlock Tree maintains the logical relationship between all the conditions (filters) in the WHERE clauses (e.g., correlated subqueries).

Then it will translate the SQL-AST to Hive AST by traversing of the input SQL-AST (as directed the QueryInfo and FilterBlock Trees). Finally, it will traverse the generated Hive AST, performing necessary optimizations and validations.

2.3 Additional stages – semantic analyzer and optimizer

As described earlier, the SQL engine will analyze and optimize the SQL-AST to generate Hive AST, so that it can reuse the semantic analyzer and optimizer in Hive when appropriate. On the other hand, some enhancement to the existing semantic analyzer and optimizer in Hive will be needed to support new SQL features (such as INTERSECT and MINUS).