Paper Summary

Spark SQL: Relational Data Processing in Spark

Major Technical Advancements in Apache Hive

Dong Chen, Huanyu Zhao, Jinxiaoyu Zhi

Problem and Motivation

The first generation of relational interface on Spark was Shark, which was the modification of Apache Hive system. The shark implemented the traditional RDBMS optimizations, such as columnar processing. Though the shark had good performance, it had three problems to be solved. First, only external data stored in the Hive catalog can be queried by Shark, therefore, the Shark can only do the relational queries on data inside a Spark program. Second, to call the Shark in Spark, putting together a SQL string is needed, however, it is pretty complex. Finally, the Hive optimizer was aimed to MapReduce, so building new features like data types for machine learning or support for new data sources were pretty hard.

Therefore, the Spark SQL was created that can let users seamlessly intermix the relational API and the procedural API, rather than forcing users to pick one of them.

Hypothesis

As there are three problems to be solved in Shark, the goals of Spark SQL is like:

1. Support relational processing both within Spark programs and on external data sources using a programmer-friendly API.
2. Provide high performance using established DBMS techniques.
3. Easily support new data sources, including semi-structured data and external databases amenable to query federation.
4. Enable extension with advanced analytics algorithms such as graph processing and machine learning.

Spark SQL bridges the gap between relational API and procedural API. First, the Spark SQL provides the DataFrame API which can accomplish relational operations both in the Spark built-in dataset and external data sources. This API evaluates operations lazily so that it can implement some optimization. Second, Spark SQL provides a novel extensible optimizer called Catalyst, which can support wide range of data sources and algorithms in big data.

Solution Overview

1. This paper provides a new module in Apache Spark called Spark SQL. Spark has the following characters:

(1) Support both relational and procedure processing. It exposes SQL interfaces, which can be accessed through JDBC/ODBC, as well as DataFrame API integrated into Spark.

(2) Provide high performance using established DBMS techniques.

(3) Easily support new data sources and algorithms.

(4) Include a highly extensible optimizer, Catalyst. Catalyst makes it easy to add data sources, optimizations rules, and data types for domains such as machine learning.

2. Below are some main components of Spark SQL:

(1) DataFrame: A DataFrame is equivalent to a table in relational database.

a. DataFrames are lazy. No execution occurs until output operation being called.

b. It provides DataFrame API to user. Unlike RDDs, DataFrame supports various relational operations that lead to optimized execution.

c. Spark SQL uses a nested data model based on Hive for tables and DataFrames.

d. DataFrame support all common relational operations, such as projection, filter, join and aggregations. Moreover, DataFrame supports User defined types (UDTS).

(2) Catalyst optimizer: Catalyst supports both rule-based and cost-based optimization.

a. Catalyst contains a library for representing trees and applying rules to manipulate them.

b. Catalyst's general tree transformations framework have four phases:

1. analyzing a logical plan to resolve references;
2. logical plan optimization;
3. physical planning;
4. code generation to compile parts of the query to Java bytecode.

c. Catalyst extension point:

1. Build a data programming model, set up s unified interface;
2. Data virtualization: Build a unified abstraction for processing heterogeneous data source.

3. Three advanced analysis features to handle challenges in “big data" environments.

(1) Schema Inference for Semistructured Data.

(2) Integration with Spark’s Machine Learning Library.

(3) Query Federation to External Databases.

4. Some advantages of DataFrames and Catalyst:

(1) Spark SQL allows developers write simpler and more efficient Spark code through the DataFrame API.

(2) DataFrame API can result in more efficient execution due to code generation.

(3) DataFrame API improves performance in applications that combine relational and procedural processing.

(4) Catalyst makes it easy to add optimization rules, data sources, and data types.

Limitations and Possible Improvements:

The performance of Spark SQL will degrade gracefully if memory is insufficient to store all RDDs. In this scenario, disk I/O overheads becomes a big issue and results in poor performance.

In the future Spark SQL releases, developers will add predicate pushdown for key-value stores such as HBase and Cassandra, which support limited forms of filtering.

Summary of class discussion:

In the class, we firstly discussed about the laziness of Dataframe and the reason: to provide more context for optimization. It is also pointed out that there is a tradeoff between instant execution and laziness, which is based on estimation of running time and resources. Besides, we also discussed about differences between dataframe and table. In the Catalyst Optimizer chapter, we exchanged our thoughts about rule base/cost base optimization difference and tradeoff. In the end, we identify issues in Mapreduce job and talked about the query optimization in Hive and pipeline optimization.