Spark SQL: Relational Data Processing in Spark

Michael Armbrust, Reynold S. Xin, Cheng Lian, Yin Huai, Davies Liu, Joseph K. Bradley, Xiangrui Meng, Tomer Kaftan, Michael J. Franklin, Ali Ghodsi, Matei Zaharia

Paper Summary by Group 5:
Alok Thatikunta (111498776), Rahul Sihag (111462160), Sweta Kumari (111497926)
{athatikunta, rsihag, swkumari} @cs.stonybrook.edu

Spark SQL is a module in Apache Spark that integrates relational processing with Spark's functional programming API. It lets the users leverage the benefits of relational processing. Compared to previous systems, it has additions like the Dataframe API for better integration with procedural Spark Code and it also includes a highly extensible language(Catalyst) to ease the addition of composable rules and control code generation. It marries imperative Spark-like data processing with declarative SQL-like data processing into a single unified interface.

It has been built on top of Spark and exposes SQL interfaces which can be accessed via Dataframe API, which lets users intermix procedural and relational language. Dataframe is a distributed collection of rows with a homogenous schema. It can be constructed from tables in system catalog or from existing Resilient Distributed Datasets(RDDs) of native Java/Python objects. Spark DataFrames are lazy unlike traditional data frame APIs, meaning no execution occurs until the user calls a special 'output operation'. It uses a nested data model based on Hive. Like Shark, Spark SQL cache the data in memory using columnar storage which is useful for interactive queries. Moreover, Spark eagerly typechecks queries even though their execution is lazy. Furthermore, Spark SQL allows users to create DataFrames of language objects (e.g. Scala objects), and UDFs are just normal Scala functions.

DataFrame queries are optimized and manipulated by a new extensible query optimizer called Catalyst. Apart from it supporting both rule based and cost based optimization, it also makes it easy for users to add new data sources and user defined types. At its core, Catalyst contains a general library for representing trees and applying rules to manipulate them. Queries are optimized in four phases, viz., Analysis in which relations and columns are resolved, queries are typechecked, etc. Then comes Logical optimization like constant folding, filter pushdown, boolean expression simplification, etc followed by Physical planning to perform Cost based optimization and then Code generation at runtime from composable expressions using Scala inbuilt quasiquote feature.

Spark SQL also added three features to handle analytical challenges. First, a schema inference algorithm for JSON and other semistructured data. It automatically infers a schema from a set of records.. Spark SQL also includes an integration with Spark's Machine Learning library, which makes it easier to perform machine learning on the data. Another utility is the query federation, allowing a single program to efficiently query disparate sources. It leverages Catalyst to do this.

Moreover, this paper also conducts an impressive experiment which shows that the performance of Spark SQL has better performance for all the task comparing to Shark and perform better in most of the case comparing to Impala. Although, the comparison with Hive and parallel databases has been missed.