ID2221 - Reading Assignment 2

Authors:

Miquel Larsson Corominas
Dmytro Siniukov

1 Summary

In the present second reading assignment, our task consists of reviewing two papers: *"Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing"* and *"Spark SQL: Relational Data Processing in Spark"*. From now on, these two papers will be reffed to in the present document as the "RDD" and the "Spark SQL" paper. The RDD paper describes a new abstraction that is designed to provide programmers and users with fault tolerant and parallelizable computations on clusters. The Spark SQL paper is focused on presenting a new powerful module to Apache Spark that provided the users with richer APIs and query optimization.

2 Motivation

The main motivation of the RDD paper was to create a system (framework) that would efficiently cope with data that is stored in memory, as all the systems that had been used previously (e.g. MapReduce) can reuse data only by storing it in an external storage, which takes a lot of computational resources. In particular, there are two main types of applications that need such a system: iterative algorithms (e.g. PageRank) and interactive data mining tools.

In the Spark SQL paper, a new module to Apache Spark is presented. Apache Spark is a well known open-source cluster computing framework, with an architecture based on RDDs which was first released in 2014, and has gained a lot of popularity in the industry thanks to its versatileness and general capabilities. This new module introduces a new abstraction called DataFrames and brings the benefits of relational processing to the Spark framework, with an included and highly configurable optimizer, named Catalyzer.

Previous to Spark SQL, effort had been made to build a interface on Spark with relational processing in a module named Shark, which had several restrictions. Basically, it could not query internal data within Spark programs, had some serious limitations that made the users work difficult and error prone, and furthermore, was not built with the need of extending the module for new data types and sources in mind, which was important for future applications like machine learning, etc.

3 Contributions

Regarding RDDs, of course, there were a lot of sub-challenges that the researchers have solved in creating the system (like the problem of representation). But, the main contribution was to develop a programming interface to handle shared memory, so that we can efficiently (and with fault tolerance) reuse intermediate results of our computations.

Regarding Spark SQL, an overview of the framework as a whole with detailed improvements is described in the next section. In general words, Spark SQL provided a solid advancement to the available tools at the moment of launch, empowering the developers analytic and computational toolbox.

4 Solution

About the RDDs, the researchers have solved the problem described above by creating an interface based on coarse-grained transformations (like map, filter, and join), which use the same kind of operation on different data items (rather than using fine-grained updates

and replicate the data to ensure the fault tolerance).

In this setting, if a piece of data has been lost, it can be easily recovered by recomputing only the current RDD, without the need to replicate the data.

In addition, the authors provide us with an open-source machinery that implements the RDD interface. It is called Spark, it is implemented in Scala programming language, and it is claimed to perform 20 times faster than Hadoop on iterative problems.

Spark SQL as a whole consists of a library that runs over Spark and provides several interfaces. The main one is the DataFrame API, which allows the users mix relational and procedural code. DataFrames are data abstractions that are structured and can be thought of equivalent to a table in a typical relational database. The difference with pure RDDs is that DataFrames support several relational operations and control its schema.

An important note about DataFrames are that they are *lazy*, meaning that the object represents a logical plan of computation, but no real processing/execution is done until commanded by the user. This paradigm allows the system to optimize computations.

Moreover, an extensible optimizer based on Scala, Catalyst, was also implemented. The important part to highlight is the fact that this tool was designed with the idea of easily allowing users to add their own customized rules and supporting new data types. At the core of the optimizer, there is a library for representing *trees* and applying *rules* to manipulate them, which provides a tree transformation framework. This framework works in four phases: Logical plan analysis, logical optimization of the plan, physical planning, and code generation (to Java bytecode) at the end.

To finalize, the paper presents us with different Spark SQL engines comparison studies, and demonstrates that the presented solution provides flexibility without compromising the overall performance.

5 Opinion

In sum, the RDD paper introduces a new flexible framework for data intensive computations. The key point of it is about performing computations on data that is located in main memory. This allows us to improve the efficiency of using (and reusing) intermediate results of computations. Moreover, the researchers have developed a system that implements the RDD framework, which is called Spark.

So, the first paper is more fundamental and provides us with the introduction of the framework, while the Spark SQL paper is about a new module that extends the programming interface with additional functionality.

The Spark SQL module is extremely powerful and allows developers and users perform more complex operations and queries in a simpler way, without compromising processing performance. On a personal note, we really enjoyed the research application section, as it gives insight into real implementations and uses. Furthermore, as we had prior experience in operating and working with Pandas libraries, we found that it was extremely easy to rapidly get used to the DSL syntax and know how to operate.