ALLUXIO

Making the Impossible Possible with Alluxio: Accelerate Spark Jobs from Hours to Seconds

Gianmario Spacagna, Harry Powell
Barclays

What's Inside

CASE STUDY

1 / Introduction

Cluster computing and Big Data technologies have enabled analysis on and insights into data. For example, a big data application might process data in HDFS, a disk-based, distributed file system. However, there are many reasons to avoid storing your on data disk, such as for data regulations, or for reducing latency. Therefore, if you need to avoid disk read/writes, you can use Spark to process the data, and temporarily cache the results in memory.

There are a number of use cases where you might want to avoid storing your data on disk in a cluster, in which case our configuration of Alluxio makes this data available in-memory in the long-term and shared among multiple applications.

However, in our environment at Barclays, our data is not in HDFS, but rather, in a conventional relational database management system (RDBMS). Therefore, we have developed an eicient workflow in Spark for directly reading from an RDBMS (through a JDBC driver) and holding this data in memory as a type-safe RDD (type safety is a critical requirement of production-quality Big Data applications). Since the database schema is not well documented, we read the raw data into a dynamically-typed Spark DataFrame, then analyze the data structure and content, and finally cast it into an RDD. But there is a problem with this approach.

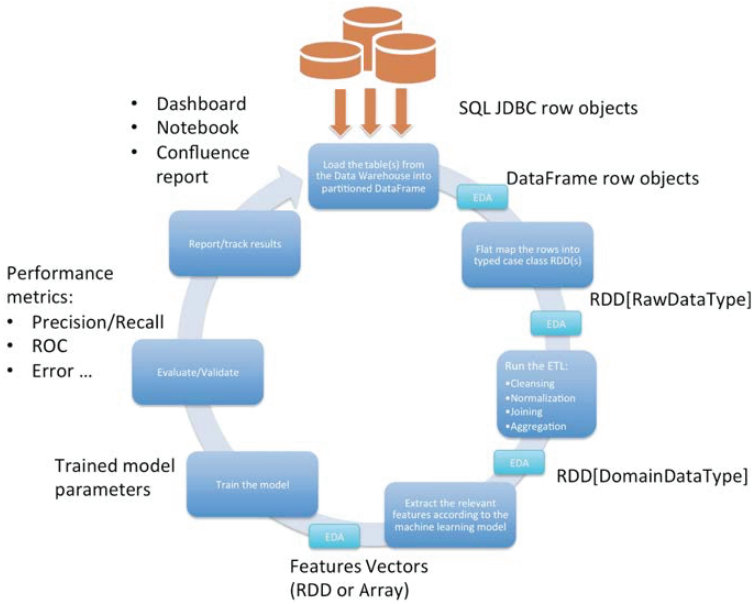
Because the data sets are large, it can take a long time to load from an RDBMS, so loading should be done infrequently. Spark can cache the DataFrame in memory, but the cached data in Spark is volatile. If we have to restart the Spark context (for example due to an error in the code, null exceptions or changes to the mapping logic) we will then have to reload the data, which could take (in our case) half an hour or more of downtime. It is not unusual to have to do this a number of times a day. Even after we have successfully defined the mapping into typed case classes, we still have to re-load the data every single time we run a Spark job, for example if there is a new feature we want to compute, a change in the model, or a new evaluation test.

Alluxio is the in-memory storage solution. Alluxio is the in-memory storage layer for data, so any Spark application can access the data in a straightforward way through the standard file system API as you would for HDFS. Alluxio enables us to do transformations and explorations on large datasets in memory, while enjoying the simple integration with our existing applications.

In this article, we first present how our existing infrastructure loads raw data from an RDBMS and uses Spark to transform it into a typed RDD collection. Then, we discuss the issues we face with our existing methodology. Next, we show how we deploy Alluxio and how Alluxio greatly improves the workflow by providing the desired in-memory storage and minimizing the loading time at each iteration. Finally, we discuss some future improvements to the overall architecture.

2 / Previous Architecture

Since the announcement of DataFrame in Spark 1.3.0 (experimental) and its evolution in recent releases (1.5.0+), the process of loading any source of data has become simple and nicely abstracted. In our case, we generate parallel JDBC connections which partition and load a relational table into a DataFrame. The DataFrame rows are then mapped into case classes. Our methodology allows us to process raw data directly from source and build our code even though the data is not physically available to the cluster disks.



3 / Issues with Existing Architecture

The main problem with our existing methodology is that the Spark cache is volatile across dierent jobs. Even though Spark provides a cache functionality, every time we restart the context, update the dependency jars or re-submit the job, the loaded data is dropped from the memory and the only way to restore it is to reload it from the central warehouse.

4 / Alluxio as the Key Enabling Technology

Alluxio is an in-memory storage system that solves our issues and enables us to take the current deployment to the next level.

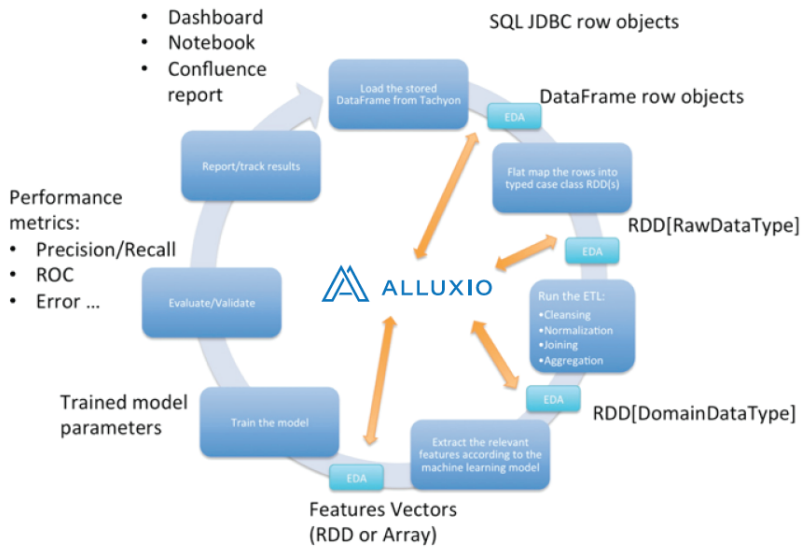
We set up Alluxio in our Spark cluster and configured no under file system (which may be Amazon S3, or HDFS, or other storage systems).

Since Alluxio is being used as an in-memory distributed file system, we can use it as storage for any text format and/or eicient data formats (such as Parquet, Avro, Kryo) together with compression algorithms (such as Snappy or LZO) to reduce the memory occupation.

To integrate with our Spark applications, we simply have to call the load/save APIs of both DataFrame and RDD and specify the path URL including the Alluxio protocol.

By having the raw data (whether it might be in Parquet-format DataFrame or Kryo-serialized Case Classes) immediately available in our Spark nodes at any time, we can now be agile and quickly iterate with our exploratory analysis and evaluation tests. We are now able to eiciently design our model and build our MVP directly from the raw source without have to face complicated and time-consuming data plumbing operations.

The following is the diagram of our workflow aer deploying Alluxio and loading the data for the first time.



5 / Making the Impossible Possible

We sped up our Agile Data Science workflow by combining Spark, Scala, DataFrame, JDBC, Parquet, Kryo and Alluxio to create a scalable, in-memory, reactive stack to explore the data and develop high quality implementations that can then be deployed straight into production.

Thanks to Alluxio, we now have the raw data immediately available at every iteration and we can skip the costs of loading in terms of time waiting, network traic, and RDBMS activity. Moreover, aer the first ETL, we save the normalized and cleaned data in memory, so that the machine learning jobs can start immediately, allowing us to run many more iterations per day.

By configuring Alluxio to keep data only in memory, the I/O cost of loading and storing into Alluxio is on the order of seconds, which in our workflow scale is simply negligible.

Our workflow iteration time decreased from hours to seconds. Alluxio enabled something that was impossible before.

The presented methodology is still an experimental workflow. Currently, there are some limitations and room for improvement. Nevertheless, we believe that the described methodology, combined with Alluxio, is a game-changer for eectively applying agile data science into large corporations.