Preparation of Papers for IEEE Computer Society TRANSACTIONS (revised November 2012)

First A. Author, Second B. Author Jr., and Third C. Author, *Member, IEEE*

Abstract—These instructions give you guidelines for preparing papers for IEEE Computer Society Transactions. Use this document as a template if you are using Microsoft Word 6.0 or later. Otherwise, use this document as an instruction set. Please note that use of IEEE Computer Society templates is meant to assist authors in correctly formatting manuscripts for final submission and does not guarantee how the final paper will be formatted by IEEE Computer Society staff. This template may be used for initial submissions; however, please consult the author submission guidelines for formatting instructions as most journals prefer single column format for peer review. An abstract should be 100 to 200 words for regular papers, no more than 50 words for short papers and comments, and should clearly state the nature and significance of the paper. Abstracts *must not* include mathematical expressions or bibliographic references. Please note that abstracts are formatted as left justified in our editing template (as shown here).

Index Terms—Keywords should be taken from the taxonomy (http://www.computer.org/mc/keywords/keywords.htm). Keywords should closely reflect the topic and should optimally characterize the paper. Use about four key words or phrases in alphabetical order, separated by commas (there should not be a period at the end of the index terms)

———————————— ◆ ————————————

1 INTRODUCTION

THIS document is a template for Microsoft Word versions 6.0 or later. If you are reading a paper version of this document, please download the electronic file from the template download page so you can use it to prepare your manuscript.

When you open the document, select “Page Layout” from the “View” menu in the menu bar (View | Page Layout), which allows you to see the footnotes. Then type over sections of the document or cut and paste from another document and then use markup styles. Please keep the template at 8.5″ x 11″—do not set the template for A4 paper. The pull-down style menu is at the left of the Formatting Toolbar at the top of your Word window (for example, the style at this point in the document is “Text”). Highlight a section that you want to designate with a certain style, and then select the appropriate name on the style menu. The style will adjust your fonts and line spacing. Use italics for emphasis; do not underline. **Do not change the font sizes or line spacing to squeeze more text into a limited number of pages. Please be certain to follow all submission guidelines when formatting an article or it will be returned for reformatting.**

To modify the running headings, select View | Header and Footer. Click inside the text box to type the name of the journal the article is being submitted to and the manuscript identification number. Click the forward arrow in the pop-up tool bar to modify the header or footer on subsequent pages.

To insert images in Word, position the cursor at the insertion point and either use Insert | Picture | From File or copy the image to the Windows clipboard and then Edit | Paste Special | Picture (with “Float over text” unchecked).

IEEE Computer Society staff will edit and complete the final formatting of your paper.

- *F.A. Author is with the National Institute of Standards and Technology, Boulder, CO 80305. E-mail: author@ boulder.nist.gov.*
- *S.B. Author Jr. is with the Department of Physics, Colorado State University, Fort Collins, CO 80523. E-mail: author@colostate.edu.*
- *T.C. Author is with the Electrical Engineering Department, University of Colorado, Boulder, CO 80309. On leave from the National Research Institute for Metals, Tsukuba, Japan E-mail: author@nrim.go.jp.*

******Please provide a complete mailing address for each author, as this is the address the 10 complimentary reprints of your paper will be sent***

*Please note that all acknowledgments should be placed at the end of the paper, before the bibliography (**note that corresponding authorship is not noted in affiliation box, but in acknowledgment section**).*

2 Programing Model

2.1 Resilient Distributed Datasets

Spark uses a very central form of abstraction that enables it to achieve a great degree of generality while having performance comparable to a specialized system, namely, this programing abstraction are the Resilient Distributed Datasets, or RDDs for short. These are fault-tolerant collections of objects partitioned across cluster that can be manipulated in parallel.

Users can apply two types of operations on RDDs, “transformations” and “action”, the former is used to create RDDs and the latter returns a result based on the RDD that this operation was executed on, we will return to this later.

RDDs are exposed through a functional programming API available in Scala, Java, Python and R. This API allows the user to simply pass local functions to run on the cluster.

2.2 RDD operations

As previously mentioned, there are two types of operations that a user can apply to RDDs, namely “transformations” and “action”, in this part we will explore further such operations.

RDD transformation, such as map, filter and groupBy, are functions that given an RDD as input returning one or more RDDs as output, effectively creating new RDDs. These transformations are lazy in nature, meaning, once called they are not computed immediately, that is Spark only mantains the record of which operation is being called, as well as the relations between RDDs through a directed acyclic graph, DAG for short, also known as the graph of transformations, in parallel Spark also creates an execution plan based in the graph of transformations, so right after creation an RDD does not contain data. After an action operation is called Sparks looks at the whole graph of transformations used to create the execution plan and finally computes the RDD.

This lazy evaluation of operations of type transformation provides several benefits, one of them is that it allows spark to find an efficient plan for the user’s computation, that is, for example, if we have several filter operations in a row Spark can merge them all into a single filter, this allows for better performance by reducing the number of queries and allows the development of modular programs without losing performance.

RDD action, like count, collect and take, are functions used to work with the actual dataset created by a RDD transformation and return a value based on the RDD that this operation was executed on. An RDD action doesn’t create a new RDD unlike the RDD transformation, so the returned values are non-RDD values that are stored to drivers or to the externel storage system. This not only brings the laziness of RDD into motion but each time a action is called on a given RDD, that RDD must be recomputed for each of the actions, this is caused because RDDs are ephemeral by default, however RDDs have explicit support for data sharing among computation, this will be explored further in the following sub chapter.

2.3 RDD data sharing

RDDs provide explicit support for data sharing among computations, meaning, users can choose to persist selected RDDs in memory for rapid reuse, this operation is achieved by calling the persist action, and if the data does not fit in memory, Spark will spill it to disk, it is also possible to choose where to store the dataset. This approach resolves the issue of needing to recompute the same RDD each time an action is called on it, providing speedups of up to 100x and being key to Spark’s generality.

When a RDD is persisted, each node saves only partitions of said RDD that it has computed in memory, and reuses in other actions on that dataset, or datasets derived from it.

2.4 RDD fault tolerance

Besides data sharing and a variety of operations, RDDs are also capable of automatically recover from failures. This is achieved through an approach named “lineage”, where in each RDD tracks the graph of transformations that was used to buid it, and if any fault arises in any machine, Spark can use this graph to rebuild any partition of the lost dataset on said machine by recomputing the RDD

3 Higher-Level Libraries

Spark offers a variety of higher-level libraries, targeting many of the use cases of specialized computing engines, such libraries are based on the RDD approach, this allows the libraries to have comparable performance with specialized systems designed with the same use case in mind as well as allowing for easy combining of the different libraries in applications, for example, combining the machine learning library with the streaming library to train an artificial intelligence using live data, or combining the machine learning library with the SQL library to train it using data store in databases, this process is made easy due to this communality of using RDDs making the development of said applications faster.

There are four main libraries that we will discuss in detail in the following sub chapters, these libraries are the following

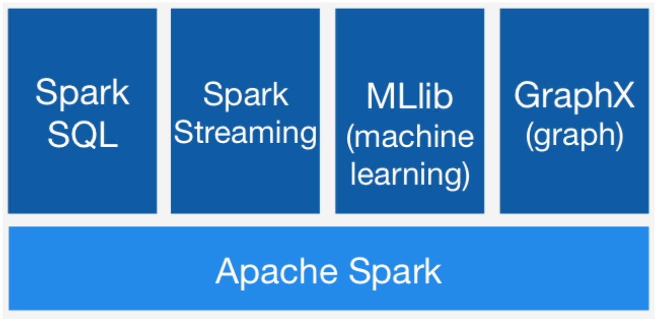

Fig. 1. The main libraries offered by Apache Spark.

3.1 Spark SQL

Spark SQL, as the name implies, is responsible for the implementation of SQL and Dataframes. Spark SQL uses techniques similar to analytical databases, that is, inside an RDD, the data layout is the same as the one used in analytical databases, that being one of compressed columnar storage, that is, each column is its own independent array, holding the values of that column for each row, and for each of these arrays is applied type specific compression, also using cost-based optimization and code generation for query execution.

Spark SQL also introduces an extra level of abstraction, that being DataFrames, these being RDDs that have a known schema, in other words, these represent a dataset organized into named columns, being conceptually equivalent to a table in a relational database or a data frame in python/R, but with richer optimizations like using Spark SQL's query planner, making it so user code receives optimizations like: predictive pushdown, operator reordering and join algorithm selection, and much like RDDs, DataFrames also execute lazily.

A DataFrame can be created from a wide variety of sources, such as: structured data files, like JSON, tables in Hive, external databases or existing RDDs. In short Spark SQL is structured like so:

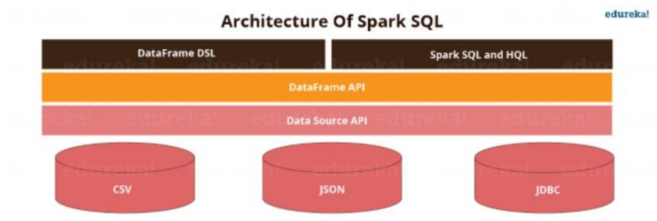

Fig. 2. Architecture of Spark SQL.

Where Data Source API is responsible for the loading and storing structured data, supporting Hive, Avro, JSON, JDBC, Parquet, and others.

The DataFrame API implements the DataFrame abstraction mentioned earlier.

One technique that is, as of time of writing, not implemented on Spark SQL is indexing, a technique that involves the use a index table that points to a block of data where a specific data entry is located speeding up the query, although other libraries such as IndexedRDDs do use it.

3.2 Spark Streaming

Using a model called "discretized streams", the Spark streaming library implements incremental, scalable, high-throughput and fault-tolerant processing of live data streams, this being the core scheduling module of Spark.

Streaming over Spark is achived by splitting the input data into small batches such as every 200 milliseconds that are combined with the state inside RDDs. This way Spark streaming can take adventage of fault recovery, namely the "lineage" approach mentioned before, this being less expensive than traditional methods, offering the low latency required to work with live data. Also offers the possibility to combine the different tools offered with Spark with Spark Streaming easing the processing of the data.

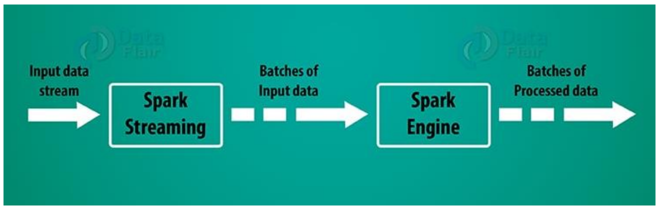

Fig. 3. Typical structure of a stream processing system in Spark.

Figure 3 represents the typical structure of a stream processecing system, where Spark Streaming receives the input data, sending it to the spark engine to be processed.

3.3 GraphX

GraphX is responsible for providing a graph computation interface similar to Pregel and GraphLab, offering the same placement optimizations as these systems, such as vertex partitioning schemes.

3.4 MLlib

This library is Spark's machine learning library, implementing over fifty common algorithms for distributed model training like common distributed algorithms of decision trees (PLANET), Latent Dirichlet Allocation, Alternating Least Squares matrix factorization.

Before Spark 2.0 the main machine learning API was RDD based, but after Spark 2.0 the main API is the DataFrame based API, these DataFrames being the same mentioned before in the Spark SQL sub section. With this MLlib includes a framework for creating machine learning pipelines, through the standardization of APIs for machine learning that MLlig offers, making it easier to combine multiple algorithms into a single pipeline or workflow. This standardization is achivied through the introduction of the ML Pipelines concept.

A ML Pipeline in MLlib provide a uniform set of high-level APIs built on top of the DataFrames API and can be divided into five main concepts: DataFrames, Transformer, Estimator, Pipeline and Parameter.

A DataFrame is the same as the one mentioned in the Spark SQL section, so it will not be explored deeply again.

Transformers are akin to transformation operation in RDDs but applied to a DataFrame, that is a transformer implements a method that converts a DataFrame into another.

Estimators are yet another form of abstraction offered by Spark, where an estimator abstracts the concept of a learning algorithm or any algorithm that fits or trains on data.

A Pipeline consists basically a sequence of transformers and estimators to be run in a specific order.

Parameters are the parameters for estimator and transformers, like for example the max number of iterations of a logistic regression, parameters can be set individually for an instance or through the passing of a paramMap, that's a set of (parameter, value) pairs.

4 APLICATIONS

Spark is used in a wide range of applications, having more than 1000 companies using it, in areas such as web services to biotechnology to finance, also having several scientific applications in academia of several scientific domains, typically combining several of the libraries offered by Spark.

4.1 Batch Processing

One of the most common applications of Spark is batch processing on large datasets, including Extract-Transform-Load workloads to convert raw data into a more structured format, and the training of machine learning models. Examples of this workload are: personalization and recommendation at Yahoo, managing a data lake at Goldman Sachs, graph mining at Alibaba, financial Value at Risk calculation, text mining of customer feedback at Toyota and a 8000 node cluster at Tencent that ingests 1PB of data per day. Applications of this type often run only on disk, instead of in memory.

4.2 Interactive queries

Interactive use of Spark can be divided into three main classes:

Spark SQL for relational queries, meaning organizations often use Spark SQL to compute interactive queries on databases, often through business intelligence tools like Tableau. Examples of this include eBay and Baidu.

Using Sparks API through shells or visual notebook environments. This is crucial for developers and data scientists for asking more advanced questions and for designing models that eventually to the production of applications.

Domain specific interactive applications that run on spark, developed by several vendors. Examples of this include Tresata, Trifacta and PanTera.

4.3 Stream processing

Another popular use case of Spark is real time processing of data, both in analytics and in real-time decision-making applications. Real world examples of use of this use case include network security and monitoring at Cisco, prescriptive analytics at Samsung SDS and log mining at Netflix.

Applications that use streaming are often combined with batch processing and interactive queries, like for example Conviva, a video company that uses Spark to maintain a model of content distribution server performance, querying it automatically when it moves clients across servers, in a application that requires substantial parallel work for both model maintenance and queries, being that this model is maintained continuously.

4.4 Scientific aplications

Spark is also used in several scientific domains including large-scale spam detection, image processing and genomic data processing. One example is Thunder, a platform for neuroscience at Howard Hughes Medical Institute, Janelia Farm, this combines batch processing, interactive queries and stream processing to process brain imaging data from experiments in real time, scaling up to 1TB/hour of whole brain imaging data from organinsm. Thunder allows researchers to apply machine learning algorithms to identify neurons involved in specific behaviours.