CrossMark

REVIEW

Big data analytics on Apache Spark

Salman Salloum[1] · **Ruslan Dautov**[1] · **Xiaojun Chen**[1] · **Patrick Xiaogang Peng**[1] ·
Joshua Zhexue Huang[1]

Abstract Apache Spark has emerged as the de facto framework for big data analytics with its advanced in-memory programming model and upper-level libraries for scalable machine learning, graph analysis, streaming and structured data processing. It is a general-purpose cluster computing framework with language-integrated APIs in Scala, Java, Python and R. As a rapidly evolving open source project, with an increasing number of contributors from both academia and industry, it is difficult for researchers to comprehend the full body of development and research behind Apache Spark, especially those who are beginners in this area. In this paper, we present a technical review on big data analytics using Apache Spark. This review focuses on the key components, abstractions and features of Apache Spark. More specifically, it shows what Apache Spark has for designing and implementing big data algorithms and pipelines for machine learning, graph analysis and stream processing. In addition, we highlight some research and development directions on Apache Spark for big data analytics.

Keywords Big data · Data analysis · Distributed and parallel computing · Cluster computing · Apache Spark · Machine learning · Graph analysis · Stream processing · Resilient Distributed Datasets

1 Introduction

Big data analytics is one of the most active research areas with a lot of challenges and needs for new innovations that affect a wide range of industries. To fulfill the computational requirements of massive data analysis, an efficient framework is essential to design, implement and manage the required pipelines and algorithms. In this regard, Apache Spark has emerged as a unified engine for large-scale data analysis across a variety of workloads. It has introduced a new approach for data science and engineering where a wide range of data problems can be solved using a single processing engine with general-purpose languages. Following its advanced programming model, Apache Spark has been adopted as a fast and scalable framework in both academia and industry. It has become the most active big data open source project and one of the most active projects in the Apache Software Foundation.

As an evolving project in the big data community, having good references is a key need to get the most of the Apache Spark and contribute effectively to its progress. While the official programming guide[1] is the most up-to-date source about Apache Spark, several books (e.g., [37,45,70]) have been published to show how Apache Spark can be used to solve big data problems. In addition, Databricks, the company founded by the creators of Apache Spark, has developed a set of reference applications[2] to demonstrate how Apache Spark can be used for different workloads. Other good sources are the official blog[3] at Databricks and Spark

✉ Salman Salloum
ssalloum@szu.edu.cn

1   Big Data Institute, College of Computer Science and
    Software Engineering, Shenzhen University, Shenzhen,
    Guangdong, China

---

1   http://spark.apache.org/docs/latest/programming-guide.html.

2   https://www.gitbook.com/book/databricks/databricks-spark-reference-applications.

3   https://databricks.com/blog/.

🖄 Springer

Hub[4] where you can find Spark's news, events, resources, etc. However, the rapid adoption and development of Apache Spark, coupled with an increasing research on using it for big data analytics, make it difficult for beginners to comprehend the full body of development and research behind it. To our knowledge, there is no comprehensive summary on big data analytics using Apache Spark.

In order to fill this gap, help in getting started with Apache Spark and follow such an active project,[5] the goal of this paper is to provide a concise succinct source of information about the key features of Apache Spark. Specifically, we focus on how Apache Spark can enable efficient large-scale machine learning, graph analysis and stream processing. Furthermore, we highlight the key research works behind Apache Spark and some recent research and development directions. However, this paper is not intended to be an in-depth analysis of Apache Spark.

The remainder of this paper is organized as follows. We begin with an overview of Apache Spark in Sect. 2. Then, we introduce the key components of Apache Spark stack in Sect. 3. Section 4 introduces data and computation abstractions in Apache Spark. In Sect. 5, we focus on Spark's MLlib for machine learning. Then, we move to GraphX for graph computation in Sect. 6. After that, we show the key features of Spark Streaming in Sect. 7. In Sect. 8, we briefly review some benchmarks for Apache Spark and big data analytics. Building on the previous sections, we highlight some key issues with Apache Spark in Sect. 9. Finally, summary and conclusions of this paper are presented in Sect. 10.

2 Overview of Apache Spark

In this first section, we introduce an overview of Apache Spark project and Spark's main components. We highlight some key characteristics which make Apache Spark a next-generation engine for big data analytics after Hadoop's MapReduce. We also summarize some contributions and case studies from the industry.

2.1 Apache Spark project

Spark was initially started by Matei Zaharia at UC Berkeley's AMPLab in 2009 and open sourced in 2010 under a BSD license. Then, the project was donated to the Apache Software Foundation in 2013. Several research projects have made essential contributions for building and improving Spark core and the main upper-level libraries [7,33,61,83,89, 90,93–95]. For example, the development of Spark's MLlib

began from MLbase[6] project, and then, other projects started to contribute (e.g., KeystoneML[7]). Spark SQL started from Shark project [84], and then, it became an essential library in Apache Spark. Also, GraphX started as a research project at the AMPLab.[8] Later, it became a part of the Apache Spark project since version 0.9.0. Many packages[9] have also been contributed to Apache Spark from both academia and industry. Furthermore, the creators of Apache Spark founded Databricks,[10] a company which is closely involved in the development of Apache Spark.

2.2 Main components and features

Apache Spark system consists of several main components including Spark core [90,93,94] and upper-level libraries: Spark's MLlib for machine learning [61], GraphX [33,83,85] for graph analysis, Spark Streaming [95] for stream processing and Spark SQL [7] for structured data processing. It is evolving rapidly with changes to its core APIs and addition of upper-level libraries. Its core data abstraction, the Resilient Distributed Dataset (RDD), opens the door for designing scalable data algorithms and pipelines with better performance. With the RDD's efficient data sharing and a variety of operators, different workloads can be designed and implemented efficiently. While RDD was the main abstraction introduced in Spark 1.0 through the RDD API, the representation of datasets has been an active area of improvement in the last two years. A new alternative, the DataFrame API, was introduced in Spark 1.3, followed by a preview of the new Dataset API in Spark 1.6. Moreover, a major release (Spark 2.0) was released at the time of writing this paper [81,91].

2.3 From Hadoop's MapReduce to Apache Spark

Apache Spark has emerged as the de facto standard for big data analytics after Hadoop's MapReduce. As a framework, it combines a core engine for distributed computing with an advanced programming model for in-memory processing. Although it has the same linear scalability and fault tolerance capabilities as those of MapReduce, it comes with a multistage in-memory programming model comparing to the rigid map-then-reduce disk-based model. With such an advanced model, Apache Spark is much faster and easier to use. It comes with rich APIs in several languages (Scala, Java, Python, SQL and R) for performing complex distributed

---

operations on distributed data. In addition, Apache Spark leverages the memory of a computing cluster to reduce the dependency on the underlying distributed file system, leading to dramatic performance gains in comparison with Hadoop's MapReduce [31]. It is also considered as a general-purpose engine that goes beyond batch applications to combine different types of computations (e.g., job batches, iterative algorithms, interactive queries and streaming) which previously required different separated distributed systems [45]. It's built upon the Resilient Distributed Datasets (RDDs) abstraction which provides an efficient data sharing between computations. Previous data flow frameworks lack such data sharing ability although it is an essential requirement for different workloads [90].

2.4 A unified engine for big data analytics

As the next-generation engine for big data analytics, Apache Spark can alleviate key challenges of data preprocessing, iterative algorithms, interactive analytics and operational analytics among others. With Apache Spark, data can be processed through a more general directed acyclic graph (DAG) of operators using rich sets of transformations and actions. It automatically distributes the data across the cluster and parallelizes the required operations. It supports a variety of transformations which make data preprocessing easier especially when it is becoming more difficult to examine big datasets. On the other hand, getting valuable insights from big data requires experimentation on different phases to select the right features, methods, parameters and evaluation metrics. Apache Spark is natively designed to handle such kind of iterative processing which requires more than one pass over the same dataset (e.g., MLlib for designing and tuning machine learning algorithms and pipelines).

In addition to iterative algorithms, Apache Spark is well suited for interactive analysis which can quickly respond to user's queries by scanning distributed in-memory datasets. Moreover, Apache Spark is not only a unified engine for solving different data problems instead of learning and maintaining several different tools, but also a general-purpose framework which shortens the way from explanatory analytics in the laboratory to operational analytics in production data applications and frameworks [70]. Consequently, it can lead to a higher analyst productivity, especially when its upper-level libraries are combined to implement complex algorithms and pipelines.

2.5 Apache Spark in the industry

Since its initial releases, Apache Spark has seen a rapid adoption by enterprises across a wide range of industries.[11]

Such fast adoption with the potential of Apache Spark as a unified processing engine, which integrates with many storage systems (e.g., HDFS, Cassandra, HBase, S3), has led to dozens of community-contributed packages that work with Apache Spark. Apache Spark has been leveraged as a core engine in many world-class companies such as IBM, Huawei [79], Tencent and Yahoo. For example, in addition to FP-growth algorithm and the Power Iteration Clustering algorithm, Huawei developed Astro[12] which provides native, optimized access to HBase data through Spark SQL/ Dataframe interfaces. With a major commitment to Apache Spark, IBM founded a Spark technology center.[13] Also, IBM SystemML[14] was open sourced and there is a plan to collaborate with Databricks to enhance machine learning capabilities in Spark's MLlib. Furthermore, Microsoft announced a major commitment[15] to support Apache Spark through Microsoft's platforms and services such as Azure HDInsight[16] and Microsoft R Server.[17]

There are considerable case studies of using Apache Spark for different kinds of applications: e.g., planning and optimization of video advertising campaigns at Eyeview [18], categorizing and prioritizing social media interactions in real time at Toyota Motor Sales, USA [50], predicting the off-lining of digital media at NBC Universal [14] and real-time anomaly detection at ING banking [15]. It is used to manage the largest computing cluster (8000+ nodes) at Tencent and to process the largest Spark jobs (1 PB) at Alibaba and Databricks [92]. Also, the top streaming intake (1.2 TB/h) using Spark Streaming for large-scale neuroscience was recorded at HHMI Janelia Farm Research Campus [30]. Apache Spark has also set a new record as the fastest open source engine for large-scale sorting[18] (1 PB in 4 h) in 2014 on disk sort record [80].

3 Apache Spark stack

Apache Spark consists of several main components including Spark core and upper-level libraries (Fig 1). Spark core runs on different cluster managers and can access data in any Hadoop data source. In addition, many packages have been built to work with Spark core and the upper-level libraries.

---

[11] Powered By Spark: http://tinyurl.com/h62q3ep.

[12] https://github.com/HuaweiBigData/astro.

[13] http://www.spark.tc/.

[14] https://developer.ibm.com/open/systemml/.

[15] Microsoft Announcement: http://tinyurl.com/gmjwan9.

[16] https://azure.microsoft.com/en-us/services/hdinsight/apache-spark/.

[17] https://azure.microsoft.com/services/hdinsight/r-server/.
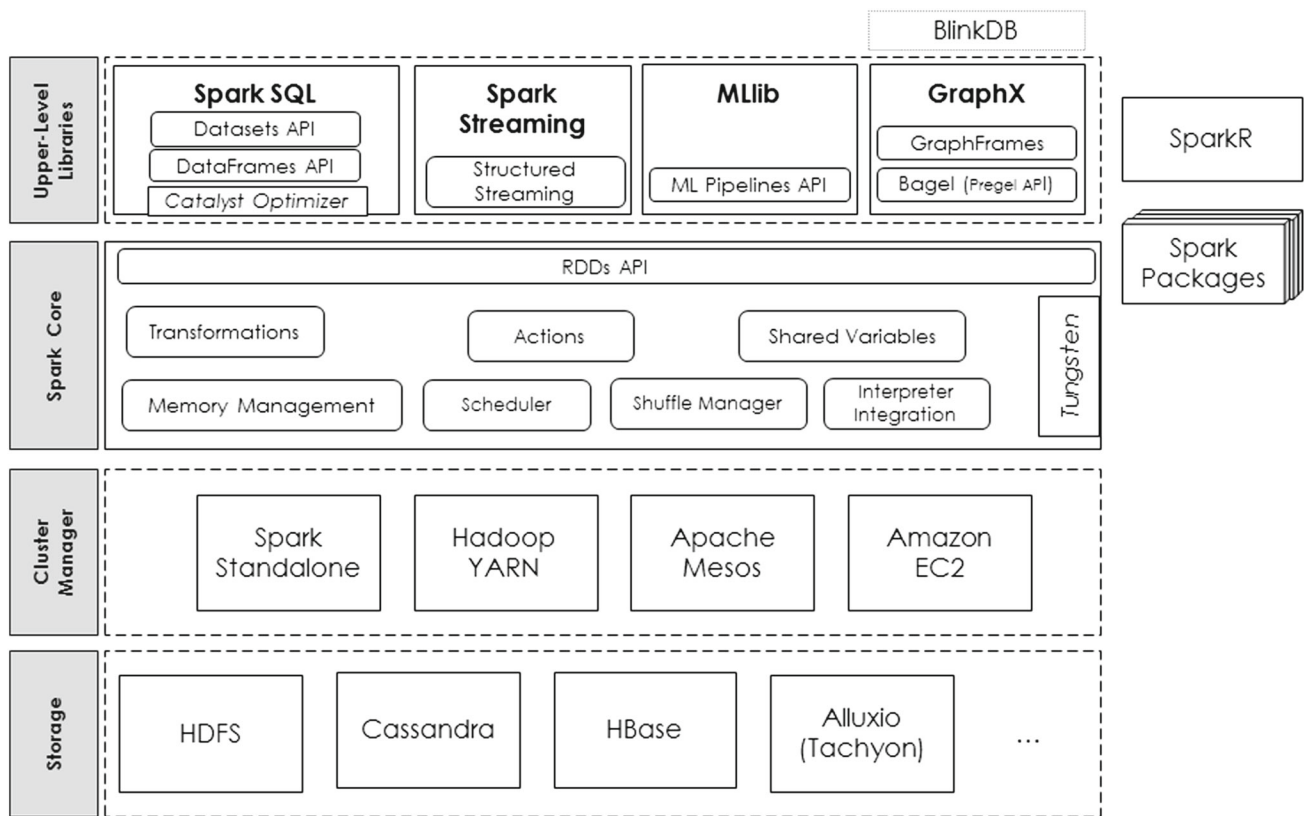
[18] http://sortbenchmark.org/.

Fig. 1 High-level architecture of Apache Spark stack

For a general overview of big data frameworks and platforms, including Apache Spark, refer to the big data landscape.[19]

3.1 Spark core

Spark core is the foundation of Apache Spark. It provides a simple programming interface for processing large-scale datasets, the RDD API. Spark core is implemented in Scala, but it comes with APIs in Scala, Java, Python and R. These APIs support many operations (i.e., data transformations and actions) which are essential for data analysis algorithms in the upper-level libraries. In addition, Spark core offers main functionalities for in-memory cluster computing including memory management, job scheduling, data shuffling and fault recovery. With these functionalities, a Spark application can be developed using the CPU, memory and storage resources of a computing cluster.

3.2 Upper-level libraries

Several libraries have been built on top of Spark core for handling different workloads: Spark's MLlib for machine learning [61], GraphX [33,83] for graph processing, Spark

Streaming [95] for streaming analysis and Spark SQL [7] for structured data processing. Improvements in Spark core lead to corresponding improvements in the upper-level libraries as these libraries are built on top of Spark core. The RDD abstraction has extensions for graph representation (i.e., Resilient Distributed Graphs in GraphX) and stream data representation (i.e., Discritized Streams in Spark Streaming). In addition, the DataFrame and Dataset APIs of Spark SQL provide a higher level of abstraction for structured data.

3.3 Cluster managers and data sources

A cluster manager is used to acquire cluster resources for executing jobs. Spark core runs over diverse cluster managers including Hadoop YARN [76], Apache Mesos [39], Amazon EC2 and Spark's built-in cluster manager (i.e., standalone). The cluster manager handles resource sharing between Spark applications. On the other hand, Spark can access data in HDFS, Cassandra.[20] HBase, Hive, Alluxio and any Hadoop data source.
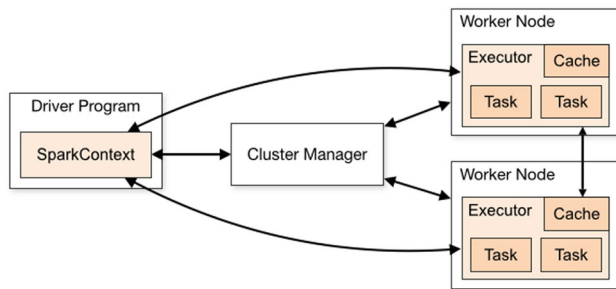
---

[19] http://mattturck.com/2016/02/01/big-data-landscape/.

[20] https://github.com/datastax/spark-cassandra-connector.

Fig. 2 Key entities for running a Spark application (*Source*: https://spark.apache.org/docs/latest/cluster-overview.html)

3.4 Spark applications

Running a Spark application involves five key entities[21] (Fig 2): a driver program, a cluster manager, workers, executors and tasks. A driver program is an application that uses Spark as a library and defines a high-level control flow of the target computation. While a worker provides CPU, memory and storage resources to a Spark application, an executer is a JVM (Java Virtual Machine) process that Spark creates on each worker for that application. A job is a set of computations (e.g., a data processing algorithm) that Spark performs on a cluster to get results to the driver program. A Spark application can launch multiple jobs. Spark splits a job into a directed acyclic graph (DAG) of stages where each stage is a collection of tasks. A task is the smallest unit of work that Spark sends to an executor. The main entry point for Spark functionalities is a *SparkContext* through which the driver program access Spark. A *SparkContext* represents a connection to a computing cluster.

3.5 Spark packages and other projects

Spark packages are open source packages and libraries that integrate with Apache Spark, but are not part of the Apache Spark project. Some packages are built to work directly with Spark core and others to work with upper-level libraries. Currently, there are more than 200 packages[22] in different categories such as: Spark core, data sources, machine learning, graph, streaming, pySpark, deployment, applications, examples and other tools.

While there are several projects which have contributed to building key components of Apache Spark project, many other projects[23] and applications are built on top of Apache Spark and its upper-level libraries. We list here some supplemental and related projects for Apache Spark:

- *MLbase* [24]: a distributed machine learning library at scale on Spark core. It has led to the current Spark's MLlib.
- *KeystoneML* [25]: a framework for large-scale machine learning pipelines on Spark core. It has contributed to the current Spark's ML pipelines API.
- *Tungsten*: fast in-memory processing for Spark applications. It is currently a key component of Spark's execution engine [82]. This will reduce the memory overhead by leveraging off heap memory. Tungsten is expected to become the de facto memory management system [31].
- *Alluxio (formerly Tachyon)* [26]: an open source memory-centric distributed storage system [52].
- *SparkR* [27]: an R package that provides a frontend to use Spark from R [77]. It is now part of Apache Spark project.
- *BlinkDB* [28]: an approximate query engine on Spark SQL[1].
- *Spark Job Server* [29]: a RESTful interface for submitting and managing Apache Spark jobs, jars and job contexts.

4 Abstractions of data and computation

Apache Spark introduces several key abstractions for representing data and managing computation. At the low-level, data are represented as Resilient Distributed Datasets (RDDs) and computations on these RDDs are represented as either transformations or actions. In addition, there are broadcast variables and accumulators which can be used for sharing variables across a computing cluster.

4.1 Resilient Distributed Datasets

Spark core is built upon the Resilient Distributed Datasets (RDDs) abstraction [94]. *An RDD is a read-only, partitioned collection of records. RDDs provide fault-tolerant, parallel data structures that let users store data explicitly on disk or in memory, control its partitioning and manipulate it using a rich set of operators* [90]. It enables efficient data sharing across computations, an essential requirement for different workloads. An RDD can be created either from external data sources or from other RDDs.

As a fault-tolerant distributed memory abstraction, RDD avoids data replication by keeping the graph of operations (i.e., an RDD's lineage—Fig. 3) that were used to construct it. It can efficiently recompute data lost on failure. The par-

---

[21] https://spark.apache.org/docs/latest/cluster-overview.html.

[22] http://spark-packages.org/.

[23] Supplemental Spark Projects: http://tinyurl.com/j4z3ppl.

[24] http://mlbase.org/.

[25] http://keystone-ml.org/.

[26] http://www.alluxio.org/.

[27] https://spark.apache.org/docs/latest/sparkr.html.

[28] http://blinkdb.org/.

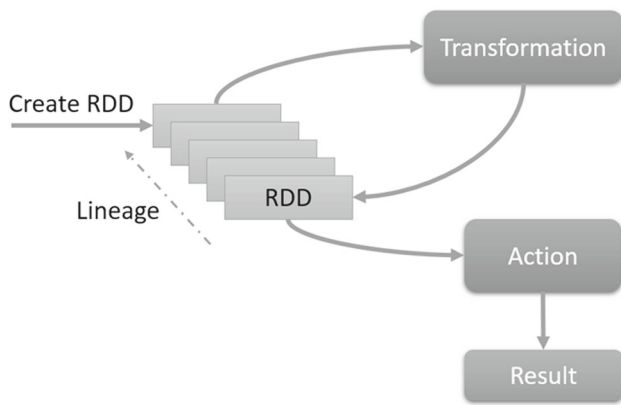[29] https://github.com/spark-jobserver/spark-jobserver.

Fig. 3 Lazy evaluation of RDDs: transformations on RDDs are lazily evaluated, meaning that Spark will not compute RDDs until an action is called. Spark keeps track of the lineage graph of transformations, which is used to compute each RDD on demand and to recover lost data (image adapted from: http://www.slideshare.net/GirishKhanzode/apache-spark-core)

Table 1 Examples of models expressible using RDDs

Category	Examples
Iterative algorithms	DryadLINQ [88], Iterative MapReduce (e.g., HaLoop [13] and Twister [26]), GraphLab [56] PowerGraph [34], Google's Pregel [58] (a Pregel API was implemented on top of Spark core [94]) and of course Spark's MLlib [61].
Relational queries	Shark [84] (currently Spark SQL [7]).
Stream processing	Discretized Streams (DStreams) [95], an RDD extension for streaming processing in Spark Streaming.
Graph processing	Resilient Distributed Graphs (RDG) [83] or property graphs, an RDD extension for graph processing in Spark GraphX.

titions of an RDD can be controlled to make it consistent across iterations where Spark core can co-partition RDDs and co-schedule tasks to avoid data movement. To avoid recomputation, RDDs must be explicitly cached when the application needs to use them multiple times.

Apache Spark offers the RDD abstraction through a simple programming interface. Each RDD is represented through a common interface with five pieces of information: partitions, dependencies, an iterator, preferred locations (data placement), and metadata about its partitioning schema. Such representation simplifies system design as a wide range of transformations were implemented without adding special logic to Spark scheduler for each one. With this representation, computations can be organized into independent fine-grained tasks. This representation can efficiently express several cluster computing models that previously required separate frameworks [90]. In addition to MapReduce model, Table 1 shows examples of models expressible using RDDs (both existing and new models).

Moreover, RDDs also enable the combination between these models for applications that require different processing types. This was a challenge with previous systems because it required different separate systems. As many parallel applications naturally perform coarse-grained operations (i.e., bulk transformations) on many records, RDDs are ideal for representing data in such applications.

4.2 Transformations and actions

In addition to the RDD abstraction, Spark supports a collection of parallel operations:[30] transformations and actions.

---

[30] https://spark.apache.org/docs/latest/programming-guide.html#rdd-operations.

Transformations are deterministic, but lazy, operations which define a new RDD without immediately computing it (Fig. 3). With a narrow transformation (e.g., *map, filter, etc*), each partition of the parent RDD is used by at most one partition of the child RDD. On the other hand, multiple child partitions may depend on the same partition of the parent RDD as a result of wide transformations (e.g., *join, groupByKey, etc*).

An action (e.g., *count, first, take, etc*) launches a computation on an RDD and then returns the results to the driver program or writes them to an external storage. Transformations are only executed when an action is called. At that point, Spark breaks the computation into tasks to run in parallel on separate machines. Each machine runs both its part of the transformations and the called action, returning only its answer to the driver program. With transformations and actions, computations can be organized into multiple stages of a processing pipeline. These stages are separated by distributed shuffle operations for redistributing data.

4.3 Shared variables

Although Spark uses a shared-nothing architecture where there is no global memory space that can be shared between the driver program and the tasks, it supports two types of shared variables for two specific use cases: broadcast variable and accumulators.

Broadcast variables are used to keep read-only variables cached on each machine (e.g., a copy of a large input dataset) rather than shipping a copy of them with tasks. Accumulators, on the other hand, are variables that workers can only add to through an associative operation and the driver can only read. They can be used to implement counters or sums.

4.4 DataFrames and Datasets or DataFrame and Dataset APIs

Spark core, and Apache Spark as a whole, is built upon the basic RDD API. However, as a rapidly evolving project, Apache Spark has introduced several improvements for its data abstraction which yield in a better computation model as well.

One of these improvements is the DataFrame API which is part of Spark SQL [7]. A DataFrame is conceptually equivalent to a table in a relational database or a data frame in R/Python, but Spark SQL comes with richer optimizations as Spark evaluates transformations lazily. It is a distributed collection of data, like RDD, but organized into named columns (i.e., a collection of structured records). This provides Spark with more information about the structure of both the data and the computation. Such information can be used for extra optimizations.

Although the RDD API is general, it provides limited opportunities for automatic optimizations because there is no information about the data structure or semantics of user functions. Moreover, the DataFrame API can perform relational operations on RDDs and external data sources and enables rich relational/ functional integration within Spark applications. DataFrames are now the main data representation in Spark's ML Pipelines API. Other Spark libraries started to integrate with Spark SQL through the DataFrame API such as GraphFrames[31] [24] for GraphX.

Another improvement is the Dataset API which is a new experimental interface added in Spark 1.6. It is an extension of the DataFrame API that provides a type-safe, object-oriented programming interface. *A Dataset is a strongly typed, immutable collection of objects that are mapped to a relational schema* [9]. The goal is to provide the benefits of RDDs with the benefits of Spark SQL's optimized execution engine (i.e., Spark's Catalyst optimizer [8]) and the Tungsten's fast in-memory encoding [82].

5 Machine learning on Apache Spark

In this section, we investigate Spark's scalable machine learning library, MLlib. We elaborate on the key features that simplify the design and implementation of machine learning algorithms and pipelines: linear algebra and statistics packages, data preprocessing, model training, model evaluation, ensemble methods and machine learning pipelines. In addition, we summarize some research highlights on large-scale machine learning.
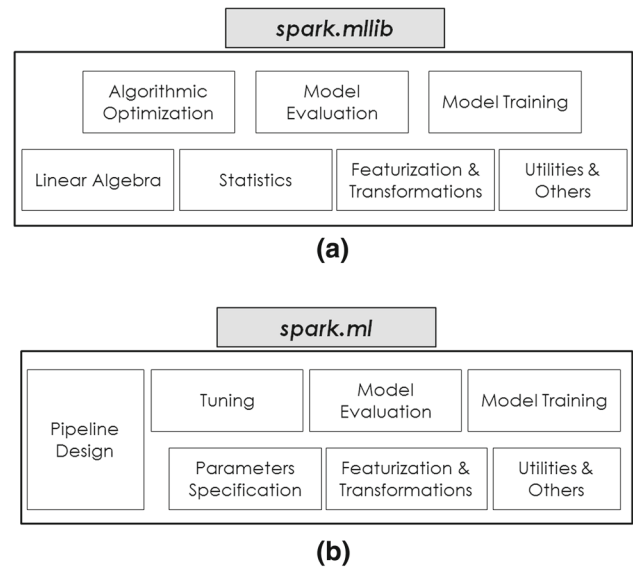
Fig. 4 Key features of Spark's MLlib: **a** *spark.mllib* is built on top of RDDs, **b** *spark.ml* is built on top of DataFrames

5.1 Spark's MLlib: key features

Apache Spark enables the development of large-scale machine learning algorithms where data parallelism or model parallelism is essential [61]. These iterative algorithms can be handled efficiently by Spark core which is designed for efficient iterative computations. Implementing machine learning algorithms and pipelines for real applications usually requires common tasks such as feature extraction, feature transformations, model training, model evaluation and tuning. In this regard, Spark's MLlib is designed as a distributed machine learning library to simplify the design and implementation of such algorithms and pipelines.

Spark's MLlib is divided into two main packages (Fig. 4): *spark.mllib* and *spark.ml*.[32] While *spark.mllib* is built on top of RDDs, *spark.ml* is built on top of DataFrames. Both packages come with a variety of common machine learning tasks such as featurization, transformations, model training, model evaluation and optimization. *spark.ml* provides the pipelines API for building, debugging and tuning machine learning pipelines, whereas *spark.mllib* includes packages for linear algebra, statistics and other basic utilities for machine learning.

5.2 Data abstraction: RDDs and DataFrames

The basic design philosophy behind *spark.mllib* is invoking various algorithms and utilities on distributed datasets represented as RDDs. However, machine learning algorithms can be applied to different data types. Thus, *spark.ml* uses

---

31 https://github.com/graphframes/.

32 https://spark.apache.org/docs/latest/ml-guide.html.

DataFrames to represent datasets. DataFrames can hold a variety of data types and provides an intuitive manipulation of distributed structured data. The schema of the data columns is known, and it can be used for runtime checking before actually running the pipeline. In addition, with DataFrames Spark can automatically distinguish between numeric and categorical features, and it can automatically optimize both storage and computation. The DataFrame API is fundamental to *spark.ml* as it also simplifies data cleaning and preprocessing by a variety of data integration functionalities in Spark SQL.

5.3 Linear algebra and statistics

In order to satisfy the requirements of distributed machine learning, the linear algebra package, *linalg* [89], provides abstractions and implementations for distributed matrices as well as local matrices and local vectors. It supports both dense and sparse representations of vectors and matrices. Sparse representation is essential in big data analytics, because sparse datasets are very common in big data due to different reasons: high-dimensional spaces, feature transformations, missing values, etc. As a result, it is usually recommended to use sparse vectors if at most 10 % of elements are nonzero [45]. This has important effects in terms of performance and memory usage.

As a distributed linear algebra library, *linalg* supports several types of distributed matrices: *RowMatrix, IndexedRow-Matrix, CoordinateMatrix* and *BlockMatrix*. On the other hand, for supervised learning algorithms (e.g., classification and regression), a training example is represented as a *LabeledPoint* which is a local vector associated with a label to represent the label and features as a data point. In addition, *spark.mllib* contains the *Rating* data type to represent ratings of products for recommendation applications.

Another important part in *spark.mllib* is the statistics packages which are essential, not only for machine learning algorithms, but also for data analytics in general. For example, the *mllib.stats* package offers common statistical functions for exploratory data analysis: summary statistics, dependency analysis (i.e., correlation), hypothesis testing, stratified sampling, kernel density estimation and streaming significance testing. In addition, there is a special package for random data generation from various distributions.

5.4 Feature extraction, transformation and selection

Defining the right features is one of the most challenging tasks in machine learning. To simplify this task, Spark's MLlib supports several methods for feature extraction, transformation and selection. While feature extraction is necessary to extract features from raw data (e.g., *TF-IDF, Word2Vec etc*), feature transformers can be used for scaling (e.g., *StandardScalar and MinMaxScalar*), normalization (e.g., *Normalizer*), converting features (e.g., *PCA*), modifying features (e.g., *Hadamard product*) and others. The library contains also some utilities for selecting subsets of features from larger sets of features (e.g., *Chi-Squared, VectorSlicer and RFormula*).

These methods are helpful if we already have the required dataset. However, sometimes it is not easy to get a real dataset. In this regard, *spark.mllib* offers several data generation methods to generate synthesized datasets for testing specific algorithms such as k-means, logistic regression, SVM and matrix factorization. In addition, the library also provides a collection of methods to validate data before applying the target algorithm (e.g., binary label validators and multilabel validators). There are also other utilities for data loading, saving and other preprocessing utilities.

5.5 Model training

The essence of any machine learning algorithm is fitting a model to the data being investigated. This is known as model training which returns a model for making predictions on new data. As Spark core has an advanced DAG execution engine and in-memory caching for iterative computations, its benefits will be evident in providing scalable implementations of learning algorithms. Spark's MLlib comes with a number of machine learning algorithms for classification, regression, clustering, collaborative filtering and dimensionality reduction.

The library comes with two major tree ensemble algorithms which use decision trees as their base models: *Gradient Boosted Trees* and *Random Forests*. In addition, *spark.ml* supports *OneVsRest* (One-vs-All), a reduction method for performing multiclass classification given a base classifier that can efficiently perform binary classification.

5.6 Model evaluation

In general, different machine learning algorithms require different evaluation metrics according to the type of application (e.g., classification, clustering and collaborative filtering). The current supported metrics can be classified into three categories: classification metrics (binary classification, multicalss classification and multilabel classification), regression metrics and ranking metrics.

5.7 Machine learning pipelines

The pipelines API, *spark.ml*, was introduced in Spark 1.2 to facilitate the creation, tuning and inspection of machine learning workflows. A machine learning workflow is represented as a *Pipeline*, which consists of a sequence of *Pipeline Stages* to be run in a specific order. Each one of these stages

can be either a *Transformer* or an *Estimator*. While a transformer is an algorithm which can transform one DataFrame into another (e.g., feature transformers and learned models), an estimator is an algorithm which can fit a DataFrame to produce a transformer (i.e., a learning algorithm or any algorithm that fits or trains on data). In addition, an evaluation stage in a machine learning workflow is represented as an evaluator which computes metrics from predictions. Both *Estimators* and *Transformers* use a uniform API for specifying parameters.

In general, there are two main phases to learn from data: a training phase where we build a model and a testing phase where we use the fitted model to make predictions on new data. In *spark.ml*, a *Pipeline* represents the training phase while the testing phase is represented as a *Pipeline Model* which is the output of a pipeline. The abstraction of pipelines and pipeline models helps to ensure that both the training and test data go through the same processing steps. As the goal of a pipeline is to fit a learning model, it is considered as an estimator. On the other hand, a pipeline model is considered as a transformer.

Pipelines and DataFrames can be used to inspect and debug machine learning workflows. In addition, complex pipelines can be built as compositions (a pipeline within a pipeline) and DAGs. Also, user-defined components can be used in pipelines.

5.8 Optimization and tuning

spark.mllib supports two main optimization methods[33]: gradient descent methods including stochastic subgradient descent (SGD) and Limited-memory BFGS (L-BFGS). Besides, *spark.ml* offers built-in parameter tuning techniques to optimize machine learning performance. It uses an optimization algorithm[34] called Orthant-Wise Limited-memory QuasiNewton (OWL-QN) [4] which is an extension of L-BFGS that can effectively handle L1 regularization and elastic net.

Another key aspect in the current implementation of machine learning algorithms is algorithmic optimization such as level-wise training and binning in training a decision tree model [2]. However, the details of such implementations are beyond the scope of this paper.

5.9 Research highlights

Advanced analytics, such as machine learning, is essential for getting valuable insights from large-scale datasets. However, it is difficult to design, implement, tune, manage and use machine learning algorithms and pipelines at scale. There

are several research projects which focus on alleviating such challenges. While some of these projects have essential contributions to Apache Spark project, others depend on Apache Spark as a core framework for solving machine learning problems.

- *MLbase*: a platform implementing distributed machine learning at scale using Spark core as a runtime engine [48]. It consists of three components: MLlib, MLI which introduces high-level ML programming abstractions for feature extraction and algorithm development, and ML Optimizer which aims to automating the construction of ML pipelines. While MLlib and MLI target ML developers, ML Optimizer targets end users. MLbase started in 2012 as a research project at UC Berkely's AMPLab. It has led to the current Spark's MLlib, but some MLbase's features are not included in Spark's MLlib. A new component called TuPAQ (Training-supported Predictive Analytic Query Planner) [74,75] automatically finds and trains models for large-scale machine learning.
- *KeystoneML*: a software framework for building and deploying large-scale machine learning pipelines with Apache Spark. KeystoneML also started as a research project[35] at UC Berkely's AMPLab. It has contributed to the design of *spark.ml*, but it includes a richer set of operators (e.g., featurizers for images, text and speech) than those currently included in *spark.ml*.
- *SystemML*[36]: a distributed and declarative machine learning platform in which ML algorithms are expressed in a higher-level language. It also provides hybrid parallelization strategies for large-scale machine learning algorithms ranging from single-node, in-memory computations, to distributed computations on Apache Hadoop and Apache Spark [12]. It started as a research project at IBM Watson Research Center to build a platform where the algorithms are compiled and executed in a MapReduce environment [32]. Open source SystemML was announced in June 2015, and it was accepted as an Apache Incubator project in November 2015.
- *Velox*: as the design of machine learning frameworks lacks the ability to serve and manage large-scale data models, Velox is one research project which aims to transform statistical models trained by Spark into data products by offering a data management system for facilitating online model management, maintenance and serving [20]. It manages the lifecycle of machine learning models, from model training on raw data to the actual actions and predictions which produce additional observations leading to further model training.

---

[33] https://spark.apache.org/docs/latest/mllib-optimization.html.

[34] https://spark.apache.org/docs/latest/ml-advanced.html.

[35] https://amplab.cs.berkeley.edu/projects/keystoneml/.

[36] http://systemml.apache.org/.

– *HeteroSpark*: in addition to building scalable machine learning frameworks, another challenge is to leverage the power of GPUs to achieve better performance and energy efficiency with applications that are both data and computation intensive such as machine learning algorithms. HeteroSpark [54] is a GPU-accelerated heterogeneous architecture integrated with Spark. It combines the massive computing power of GPUs and scalability of CPUs and system memory resources.

– *Splash*[37]: a general framework built on Apache Spark for parallelizing stochastic learning algorithms on multinode clusters [96]. It can be substantially faster than existing data analytics packages built on Apache Spark. Stochastic algorithms are efficient approaches for solving machine learning and optimization problems with large-scale datasets, parallelizing these algorithms is a key challenge, especially that they are generally defined as sequential procedures.

There are also other works which focus on implementing and testing machine learning algorithms and utilities on Apache Spark such as parallel subspace clustering [98], decision trees for time series [71], among others. However, there are other existing open source frameworks, in addition to MLlib, for machine learning with big data such as Mahout, H2O and SAMOA. As these tools have advantages and drawbacks, and many have overlapping features, deciding on which framework to use is not easy. In this regard, several papers provide comparisons between some of these tools including MLlib [51,69].

6 Graph analysis on Apache Spark

In this section, we focus on GraphX, an upper-level library for scalable graph analysis. We introduce its key features that simplify the design and implementation of graph algorithms and pipelines: graph data representation, graph operators, graph algorithms, graph builders and other utilities. In addition, we summarize some research highlights in large-scale graph analytics.

6.1 GraphX: key features

GraphX combines the advantages of both previous graph-parallel systems and current Spark's data-parallel framework to provide a library for large-scale graph analytics [83]. With an extension to the RDD API, GraphX offers an efficient abstraction for representing graph-oriented data. In addition, it comes with various graph transformations, common graph algorithms and a collection of graph builders (Fig. 5). It also
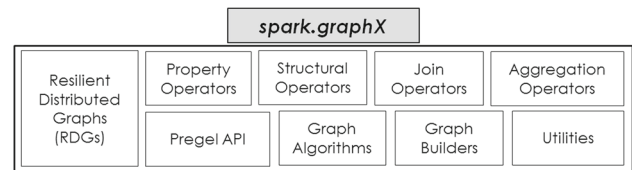
37 http://zhangyuc.github.io/splash/.

Fig. 5 Key features of GraphX

includes a variant of the Pregel API for graph-parallel computations. With GraphX, both data transformations from Spark core and graph transformations can be used. Thus, it provides an integrated framework for complete graph analysis pipelines which consist of both data and graph computation steps.

6.2 Data abstraction: RDGs

GraphX introduces the Resilient Distributed Graph (RDG) [83], an extension of the RDD API for graph abstraction. The core data structure is a property graph. A property graph is a directed multigraph (i.e., contains pairs of vertices connected by two or more parallel edges) with data attached to its vertices and edges. In other words, each vertex and each edge have properties or attributes. Like RDDs, property graphs are immutable, distributed and fault tolerant. Transformations are defined on graphs and each operation yields a new graph for changes in values and/or structure.

There are five data types for working with property graphs:

– *Graph*: an abstraction of a property graph which is conceptually equivalent to a pair of typed RDDs: one RDD is a partitioned collection of vertices and the other one is a partitioned collection of edges.
– *VertexRDD*: a distributed collection of vertices in a property graph. Each vertex is represented by a key–value pair, where the key is a unique id and the value is the data associated with the vertex.
– *Edge*: an abstraction of a directed edge in a property graph. Each edge is represented by a source vertex id, destination vertex id and edge attributes.
– *EdgeRDD*: a distributed collection of the edges in a property graph.
– *EdgeTriplet*: a combination of an edge and the two vertices that it connects. A collection of these triplets represents a tabular view of a property graph.

At the end, graph data are represented as a pair of typed RDDs. Therefore, those RDDs can be transformed and analyzed using the basic RDD API. In other words, the same graph data can be accessed and processed as a pair of collections or as a graph. Analogous to Spark core where the essential data structure is an immutable RDD (i.e., a

collection of data), the core data structure in GraphX is an immutable RDG (i.e., a graph).

6.3 Graph operators

In order to make graph analysis more flexible, GraphX extends Spark operators with a collection of specialized operators for property graphs [35]. Those operators produce new graphs with transformed properties or structure. However, the cost of such transformations is reduced by reusing substantial parts of the original graph (e.g., unaffected structure, properties and indices). The supported graph operators can be classified into the following main categories:

– *Property Operators*: There are three property transformation operators: *mapVertices*, *mapEdges* and *mapTriplets*. The graph structure is not affected by these operators and allows the resulting graph to reuse the structural indices of the original graph.
– *Structural Operators*: Current structural transformation operators include *reverse*, *subgraph*, *mask* and *groupEdges*. The associated data are not affected by the structural operators.
– *Join Operators*: *joinVertices* and *outerJoinVertices* can be used to join data from other RDDs with graphs in order to update existing properties or add new ones.
– *Aggregation Operators*: with these operators, data can be aggregated from a vertex's neighborhood. This is essential to many graph algorithms such as PageRank. *aggregateMessages* is the core aggregation operator which aggregates values for each vertex from neighboring vertices and connecting edges. Other aggregation operators compute the degree of each vertex (*maxInDegree*, *maxOutDegree*, *maxDegrees*) or collect neighboring vertices and their attributes at each vertex (*collectNeighborIds*, *collectNeighbors*).

In addition, GraphX also supports graph-parallel operators which can be used to implement custom iterative graph algorithms such as *pregel* (supported through GraphX's Pregel API). The *pregel* operator with other operators (e.g., *aggregateMessages*) can be used for implementing custom graph algorithms with a few lines of code. Moreover, GraphX comes with built-in implementations of several graph algorithms which will be reviewed briefly in the following section.

6.4 Graph algorithms

GraphX built-in algorithms include:

– *PageRank*: In order to measure the importance of each vertex in a graph, GraphX comes with two implemen-

tations of PageRank algorithm. While *staticPageRank* runs for a fixed number of iterations using the RDG API, dynamic *pageRank* uses the Pregel API and runs until the ranks stop changing.
– *Connected Components*: The *connectedComponents* finds the connected component membership for each vertex.
– *Strongly Connected Components*: The *stronglyConnectedComponents* finds the strongly connected component for each vertex and returns a graph. A strongly connected component of a graph is a subgraph containing vertices that are reachable from every other vertex in the same subgraph.
– *Triangle Counting*: In order to find the number of triangles passing through each vertex, the *triangleCount* method checks if a vertex has two neighbor vertices with an edge between them, and returns a graph in which a vertex's property is the number of triangles containing it.
– *Label Propagation*: This algorithm can be used for detecting communities in networks.
– *SVDPlusPlus*: An implementation of SVD++ based on [47], which is an integrated model between neighborhood models and latent factor models.
– *Shortest Paths*: This finds shortest paths from each vertex in a graph to a given set of vertices.

6.5 Graph builders and generators

GraphX comes with some utilities for building graph-oriented datasets from a collection of vertices and edges in an RDD or on disk. *GraphLoader* provides utilities for loading graphs from files (e.g., edge list formatted file). On the other hand, there are other methods for creating graphs from RDDs: creating a graph from RDDs of vertices and edges, creating a graph from only an RDD of edges, and creating a graph from only an RDD of edge tuples.

However, when testing graph algorithms and pipelines, it could be difficult to find the required real data with certain qualities. To alleviate this problem, GraphX offers the *GraphGenerators* utility which contains random edges generator and several other generators for generating specific types of graphs such as log normal graph (a graph whose vertex out degree distribution is log normal), R-MAT graph [17], grid graph and star graph.

6.6 Research highlights

Graph analytics, like machine learning, is also essential for getting valuable insights from large-scale graph data. A key need nowadays is a reliable framework to simplify the design and implementation of complex graph algorithms and pipelines. We list here some research directions on large graph analysis:

– *Graph-Parallel Frameworks*: Several large-scale graph-parallel frameworks were designed to efficiently execute graph algorithms, such as GraphLab [55], Pregel [57], Kineograph [19] and PowerGraph [34]. However, these frameworks have different views on graph computation and lack effective functionalities of ETL, the key challenges in big graph mining, too. Also, they offer limited fault tolerance and support for interactive data mining. GraphX, on the other hand, has a fault-tolerant, distributed graph computation, and it enables ETL and interactive analysis of massive graphs as it is built on top of Spark core and offers new graph abstraction. This can simplify the design, implementation and application of complex graph algorithms and pipelines. For more information about existing frameworks and techniques for big graph mining, refer to works such as [6].

– *Querying Big Graphs*: GraphFrames [24] integrates pattern matching and graph algorithms with Spark SQL to simplify the graph analysis pipeline and enable optimizations across graph and relational queries. This unifies graph algorithms, graph queries and DataFrames. In addition, Portal [62,63] is a declarative language built on top of GraphX to support efficient querying and exploratory analysis of evolving graphs. Another example is Quegel [86], a distributed system for large-scale graph querying.

– *Graph-based Representation*: It is clear that a reliable representation (e.g., RDGs or property graphs) of graph-oriented data is essential for efficient processing of large graphs. There are other works in this direction such as MedGraph [44] which presents a graph-based representation and computation for large sets of images.

7 Stream processing on Apache Spark

In this section, we focus on Spark Streaming, an upper-level library for large-scale stream processing. We elaborate on some key features and components: stream data abstraction, data sources and receivers, streaming computational model. We also review some examples of how Spark Streaming can be used with other Spark libraries. Then, we summarize some research highlights.

7.1 Spark Streaming: key features

Most traditional stream processing systems are designed to process records one at a time. This is known as the continuous operator model, a simple model which works very well at small scales, but it faces some challenges with large-scale and real-time analysis. In order to alleviate these challenges, Spark Streaming uses a micro-batch architecture [45] where a stream is treated as a sequence of small batches of data
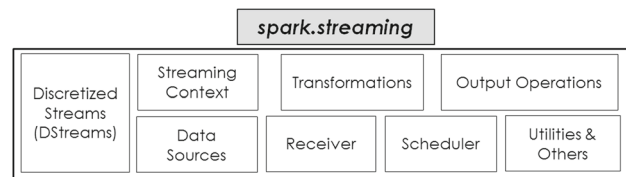

Fig. 6 Key features of Spark Streaming

and the streaming computation is done through a continuous series of batch computations on these batches of data.

To achieve such a micro-batch architecture, Spark Streaming comes with several packages or components (Fig. 6) for stream processing. The Streaming Context is the main entry point for all streaming functionality. For a Streaming Context, one parameter (the batch interval) must be set based on the latency requirements of the target application and available cluster resources. DStream is the basic programming abstraction in Spark Streaming for micro-batch stream processing. Data sources represent different kinds of streaming sources which Spark Streaming can be linked to. A receiver is like an interface, between a data source and Spark Streaming, which gets data and stores it in Spark's memory for later processing. A scheduler provides listener interface for receiving information about an ongoing streaming computation such as receiver status and processing times. In addition, Spark Streaming supports a variety of transformations, output operations and utilities for stream processing.

7.2 Data abstraction: DStreams

The basic programming abstraction in Spark Streaming is Discretized Streams (DStreams) [95]. DStream is a high-level abstraction which represents a continuous stream of data as a sequence of small batches of data. Internally, a DStream is a sequence of RDDs, each RDD has one time slice of the input stream, and these RDDs can be processed using normal Spark jobs. As a result, DStreams have the same fault tolerance properties as those of RDDs and streaming data can be processed using Spark core and other upper-level rich libraries. RDD abstraction itself is a convenient way to design the computations on data as a sequence of small, independent steps [70]. In this way, computations can be structured as a set of short, stateless, deterministic tasks instead of continuous, stateful operators. This can avoid problems in traditional record-at-a-time stream processing.

7.3 Data sources and receivers

Input DStreams are DStreams representing the stream of input data received from streaming sources. Each input DStream (except for those coming from file streams) is associated with a receiver.

7.3.1 Streaming sources

Spark Streaming provides two main categories of built-in streaming sources (basic sources and advanced sources) in addition to custom sources.

- *Basic Sources*: these sources are directly available in Spark Streaming API: file systems, socket connections and Akka[38] actor streams. In addition, a DStream can be created based on a queue of RDDs which is usually used for testing a Spark Streaming application with test data.
- *Advanced Sources*: these sources require extra packages for interfacing with external non-Spark libraries (e.g., Kafka, Flume and Twitter).
- *Custom Sources*: Spark Streaming supports creating input DStreams from custom data sources by implementing a user-defined receiver[39] that is customized for receiving data from the target data source.

7.3.2 Receivers

As its name implies, a receiver gets the data from a streaming source and stores it in Spark's memory for processing. As data sources can be reliable (i.e., allow system receiving data to acknowledge the received data correctly) or unreliable, there are two kinds of receivers as well. A reliable receiver correctly sends an acknowledgment to a reliable source when the data have been received and stored in Spark with replication. On the other hand, an unreliable receiver does not send acknowledgment to a source. However, unreliable receivers can be used for sources that do not support acknowledgment or for those which do. The *receiver* package has an interface which can be run on worker nodes to receive external data. In addition, there are several packages which provide Spark Streaming receivers for advanced sources such as Kafka,[40] Flume,[41] Kinesis,[42] Twitter[43]

7.4 Discretized stream processing

With the micro-batch architecture, Spark Streaming can receive data from different sources and divide it into small batches. New input batches are created at regular time inter-

vals (i.e., batch interval parameter). After getting data from a streaming source and storing it in Spark's memory, Spark core is used as a batch processing engine to process each batch of data. As the computation model behind Spark Streaming is based on DStreams, input data streams are discretized into batches and represented in a DStream which is stored as a sequence of RDDs in Spark's memory. Then, streaming computations are executed by generating Spark jobs to process those RDDs. This yields in other RDDs representing program outputs or intermediate states. The results of such processing can be pushed out to external systems in batches too. In order to achieve this computational model, Spark Streaming depends on the following:

7.4.1 Transformations

There are two categories of transformations on DStreams: stateless and stateful. Stateless transformations (i.e., normal RDD transformations) of each batch do not depend on the data of its previous batches. On the other hand, stateful transformations (i.e., based on sliding windows and on tracking state across time) use data or intermediate results from previous batches to compute the results of the current batch. In addition, stateless transformations are applied on each batch separately (i.e., each RDD) in a DStream. In other words, they are simple RDD transformations that apply to data within each time slice, but not across time slices. However, stateful transformations are operations which can be applied on data across time and can be divided into two main types: windowed transformations and *updateStateByKey* transformations.

Windowed transformations combine results from multiple batches. As the name indicates, these transformations can compute results within a window (i.e., within a longer time period than the batch interval). These transformations require two main parameters: *window duration* which controls how many previous batches are considered and *sliding duration* which controls how frequently the new DStream computes results (i.e., the interval at which the window operation is performed). On the other hand, an *updateStateByKey* transformation is useful to maintain a state across the batches in a DStream while continuously updating it with new information. For example, if we need to keep track of the last 10 pages each user visited on a Web site, our state object will be a list of the last 10 pages, and we will update it upon each event (i.e., accessing a web page).

7.4.2 Output operations

The actual execution of all the DStream transformations is triggered by output operations (similar to normal RDD actions). With these operations, we can specify what should be done with the final results of a stream processing, the out-

---

[38] http://akka.io/.

[39] https://spark.apache.org/docs/latest/streaming-custom-receivers.html.

[40] https://spark.apache.org/docs/latest/streaming-kafka-integration.html.

[41] https://spark.apache.org/docs/latest/streaming-flume-integration.html.

[42] https://spark.apache.org/docs/latest/streaming-kinesis-integration.html.

[43] https://dev.twitter.com/docs/streaming-apis.

put DStreams. For example, printing the results is usually used for debugging. In addition, there are other output operations for pushing the results to an external storage, such as saving them as text files or Hadoop files. Moreover, Spark Streaming supports a generic output operator, *foreachRDD*, which applies a function to each RDD on the DStream.

7.4.3 Backpressure

A mechanism that allows Spark Streaming to dynamically control the rate of receiving data when the system is in an unstable state or the processing conditions change (e.g., a large burst in input and a temporary delay in writing output). Spark Streaming started to support this mechanism in Spark 1.5.

7.4.4 DStream checkpointing

If an executer fails, tasks and receivers are restarted by Spark automatically. However, if the driver fails, Spark Streaming recovers the driver by periodically saving the DAG of DStreams to fault-tolerant storage. Then, the failed driver can be restarted from the checkpoint information.

7.5 Batch, streaming and interactive analytics

As we discussed before, Apache Spark provides a single engine for batch, streaming and interactive workloads. This makes it unique comparing to traditional streaming systems, especially regarding fast failure, straggler recovery and load balancing. In addition, Apache Spark can be used for applications which work with both streaming and static data taking advantage of the native support for interactive analysis and native integration with advanced analysis upper-level libraries. As a DStream is just a series of RDDs, the basic data abstraction for a distributed dataset in Spark, Spark Streaming has the same data abstraction with Spark core and other Spark libraries. This allows unification of batch, streaming and interactive analysis. Thus, it can simplify building real-time data pipelines which is a crucial need in many domains to get real-time insights. In the following subsections, we list some examples of using Spark Streaming with other Spark libraries.

7.5.1 Spark Streaming and Spark SQL

As RDDs generated by DStreams can be converted to DataFrames, SQL can be used to query streaming data. Some Spark reference applications [23] demonstrate how different Spark libraries can be used together. For example, log analysis application uses both Spark SQL and Spark Streaming.

7.5.2 Spark Streaming and MLlib

There are two main cases where Spark Streaming and MLlib can be used together. Machine learning models generated offline with MLlib can be applied on streaming data (i.e., Offline training, online prediction). On the other hand, machine learning models can be trained from labeled data streams (i.e., Online training and prediction). One reference application which uses Spark Streaming with Spark MLlib is Twitter Streaming Language Classifier [23]. Another one is a platform for large-scale neuroscience [28] at HHMI Janelia Farm Research Campus where Spark Streaming is integrated with MLlib to develop streaming machine learning algorithms and perform analyses online during experiments. In addition, Spark MLlib supports some streaming machine learning algorithms such as Streaming Linear Regression and Streaming K-means [29].

7.5.3 Spark Streaming and GraphX

One example of using Spark Streaming with GraphX is dynamic community detection [40] where Spark Streaming is used for online incremental community detection and GraphX is used for offline daily update. GraphTau [43] is a time-evolving graph processing framework built on top of GraphX and Spark Streaming to support efficient computations on dynamic graphs.

7.6 Research highlights

Spark Streaming is considered as one of the most widely used libraries in Spark [22]. As streaming analysis is essential in today's big data industry, it is necessary to have a reliable framework for building end-to-end analysis pipelines which integrates streaming with other workloads. In addition to the examples listed in the previous section, we list here some recent endeavors in this direction:

– *StreamDM*[44]: an open source data mining and machine learning library developed at Huawei Noah's Ark Lab and designed on top of Spark Streaming for big data stream learning.
– *IncApprox*: a data analytics system based on Spark Streaming. It combines incremental and approximating computing for real-time processing over the input data stream and emits the query result along with the confidence interval or error bounds [49].

There are other projects for stream data processing. For example, Apache Flink[45] is another open source project for

---

[44] http://huawei-noah.github.io/streamDM/.

[45] https://flink.apache.org/.

distributed stream and batch data processing. Flink started as a fork of the Stratosphere[46] research project. It became an Apache Incubator project in March 2014 and then was accepted as an Apache top-level project in December 2014. Another project is Apache Storm,[47] an open source distributed real-time computation system. A comparison between Spark streaming and other projects for stream processing is beyond the scope of this paper.

8 Benchmarks for Apache Spark

With such a rapidly evolving big data framework, reliable and comprehensive benchmarks are essential to reveal Spark's real efficiency with different workloads. The following is a summary of some works in this direction which includes benchmarking for Apache Spark and big data in general.

- *SparkBench* [53]: a Spark benchmarking suite from IBM TJ Watson Research Center [48] which covers different workloads on Apache Spark.
- *HiBench*[49]: a big data microbenchmark suite from Intel to evaluate big data frameworks such as Hadoop's MapReduce and Apache Spark.
- *Spark-perf*[50]: a performance testing framework for Apache Spark from Databricks.
- *BigBench*[51]: a specification-based benchmark for big data. It was recently used to evaluate Spark SQL [42].
- *Yahoo Streaming Benchmarks*[52]: benchmarks of three stream processing frameworks at Yahoo[53]: Apache Flink, Apache Spark and Apache Storm.
- *Spark SQL Performance Tests*[54]: a performance testing framework from Databricks for Spark SQL in Apache Spark 1.6+.
- *BigDataBench*[55]: a benchmark suite from the Chinese Academy of Sciences for evaluating different workloads using Apache Spark and other frameworks.
- *Spark Performance Analysis*[56]: a project for quantifying performance bottlenecks in distributed computation

frameworks, and using it to analyze Spark's performance on different benchmarks [66].

While some of these works are technology agnostic benchmarks (e.g., BigDataBench), others are technology-specific benchmarks which focus on Spark or some of its components (e.g., SparkBench).

9 Discussion

Currently, Apache Spark is adopted and supported by both academia and industry. The community of contributors is growing around the world, and dozens of code changes are made to the project everyday. A major release of Apache Spark (Spark 2.0) was released while writing this paper [81,91]. This paper provides a concise summary about Apache Spark from both research and development point of views. The key features of Apache Spark and the variety of applications which can be developed using this framework are clearer now. For those who want to start developing Spark applications or trying some sample programs, the Databricks community edition [57] is one place to go. However, as Apache Spark is becoming the de facto standard for big data analytics, it is also important to understand the key differences from the previous Hadoop's MapReduce model, and important research and development directions, as well as related challenges. These issues are discussed briefly in this section.

9.1 In-memory big data processing

It is clear that in-memory data abstraction is fundamental in Spark core and all its upper-level libraries, which is a key difference from the disk-based Hadoop's MapReduce model. It allows storing intermediate data in memory instead of storing it on disks and then retrieving it from disks for the subsequent transformations and actions. However, this makes memory a precious resource for most workloads on Apache Spark. On the other hand, although Apache Spark offers a flexible and advanced DAG model after the simple map/reduce model, scheduling of spark jobs is much more difficult than MapReduce jobs. Furthermore, Apache Spark is more sensitive to data quality as accessing data from remote memory is more expensive than accessing data from remote disks [53]. That is why data partitioning requires careful settings for complex applications. Moreover, optimizing shuffle operations is essential as these operations are expensive. For example, in a benchmark test using SparkBench [53], the majority workloads required more than 50 % of the total execution time for shuffle tasks.

46 http://stratosphere.eu/.

47 https://storm.apache.org/.

48 https://github.com/SparkTC/spark-bench.

49 https://github.com/intel-hadoop/HiBench.

50 https://github.com/databricks/spark-perf.

51 https://github.com/intel-hadoop/Big-Bench.

52 https://github.com/yahoo/streaming-benchmarks.

53 https://yahooeng.tumblr.com/post/135321837876.

54 https://github.com/databricks/spark-sql-perf.

55 http://prof.ict.ac.cn/BigDataBench/.

56 https://kayousterhout.github.io/trace-analysis/.

57 https://community.cloud.databricks.com/.

9.2 Data analysis workloads

Apache Spark has another key advantage which is supporting a wide range of data applications such as machine learning, graph analysis, streaming and structured data processing. While Apache Spark offers a single framework for all these workloads, different frameworks and platforms were needed for data processing with the Hadoop's MapReduce model. In addition, some of the main projects (e.g., MLbase, KeystoneML), which contributed to Spark's MLlib and ML libraries, have more features which have not yet included as part of the official releases of Apache Spark. Although Spark Streaming has been improved a lot recently, for truly low-latency, high-throughput applications, Spark may not necessarily be the right tool unless the new Structured Streaming feature is practically proved to be efficient. For detailed comparisons between Spark Streaming and other stream analysis frameworks (e.g., Apache Flink), refer to recent works such as [59]. In this regard, it's also worth noting that Apache Spark was fundamentally designed for batch processing (i.e., ETL operations).

9.3 APIs and data abstraction

Apache Spark provides easy to use APIs for operating on large data sets across different programming languages (Scala, Java, Python and R) and with different levels of data abstraction. This makes it easier for data engineers and scientists to build data algorithms and workflows with less development efforts. There are three main sets of APIs in Apache Spark, but with different levels of abstraction. Two of these APIs, the DataFrame and Dataset APIs, have been recently merged in one API in Spark 2.0.[58] This will help in unifying data processing capabilities across the upper-level libraries. The RDD API will remain the low-level abstraction which is the best choice for having a better control of low-level operations, especially when working with unstructured data. However, RDDs cannot take advantages of Spark's advanced optimizers (i.e., catalyst optimizer and Tungsten execution engine) and do not infer the schema of structured data. It is recommended to use the DataFrame and Dataset APIs when working with structured and semi-structured data. These APIs are built on top of Spark SQL which uses the Catalyst optimizer (to generate an optimized logical and physical query plan) and the Tungsten fast in-memory encoding. For a better type safety at compile time, the Dataset API is the best choice.

DataFrames and Datasets are essential for other libraries such as ML pipelines and the new Structured Streaming (i.e., Streaming DataFrames) and GraphFrames APIs. The Structured Streaming engine is developed as a core component in Spark 2.0. It is a declarative API that extends DataFrames and Datasets. With this high-level, SQL-like API, various analytic workloads (e.g., ad hoc queries and machine learning algorithms) can be run directly against a stream, for example state tracking using a stream and then running SQL queries, or training a machine learning model and then updating it. On the other hand, GraphFrames is a new API which integrates graph queries and graph algorithms with Spark SQL (or, in other words, with the DataFrame API) [24,25]. One key component of GraphFrames is a graph-aware query planner. GraphFrames are to DataFrames as RDGs are to RDDs.

All that said, choosing the right API to use may also depend on the programming language. For example, while the Dataset API is designed to work equally well with both Java and Scala, the DataFrame API is very Scala-centric. In addition, since R and Python have no compile-time type safety, the DataFrame API is suitable when working with these languages [21]. There is no doubt that data abstraction has been improved recently in Apache Spark, but having those different levels of abstractions with frequent updates may mislead developers especially when working with production applications. We believe that those APIs still need time to mature and prove their efficiency on real big data applications.

9.4 Tungsten project for memory management

While the DataFrame and Datasets APIs make Spark more accessible, the Tungsten project [82] aims to make Apache Spark faster by improving the usage efficiency of memory and CPU for Spark applications. This is essential for big data processing especially when CPU is increasingly becoming the performance bottleneck in data analysis frameworks [66]. Apache Spark included some features from the Tungsten project since Spark 1.4. Currently, the Tungsten engine is one of the core components in Spark 2.0. It is built upon ideas from modern compilers and Massively Parallel processing (MPP) databases [81].

9.5 Debugging of Spark applications

Although Apache Spark has evolved as a replacement for MapReduce by creating a framework to simplify the difficult task of writing parallelizable programs, Spark is not yet a perfectly engineered system [31]. A crucial challenge in such a framework for large-scale data processing is debugging. Developers need to understand the internals of Spark engine, the low-level architecture, in order to better identify the root causes of application failures. One recent work on this problem is the BigDebug[59] project which aims to provide a set of interactive, real-time debugging primitives for

---

[58] https://spark.apache.org/releases/spark-release-2-0-0.html.

[59] https://sites.google.com/site/sparkbigdebug/.

frameworks like Apache Spark [38]. An essential part of this project is Titian [41], a data provenance library for tracking data through transformations in Apache Spark.

9.6 Related research

In addition to the research highlights we presented in the previous sections, there are other research works which have been done using Apache Spark as a core engine for solving data problems in machine learning and data mining [5,36], graph processing [16], genomic analysis [60,65], time series data [71], smart grid data [73], spatial data processing [87], scientific computations of satellite data [67], large-scale biological sequence alignment [97] and data discretization [68]. There are also some recent works on using Apache Spark for deep learning [46,64]. CaffeOnSpark is an open source project[60] from Yahoo[61] for distributed deep learning on big data with Apache Spark.

Other works compare Apache Spark with other frameworks such as MapReduce [72], study the performance of Apache Spark for specific scenarios such as scale-up configuration [10], analyze the performance of Spark's programming model for large-scale data analytics [78] and identify the performance bottlenecks in Apache Spark [66] [11]. In addition, as Apache Spark offers language-integrated APIs, there are some efforts to provide the APIs in other languages. Mobius[62] (formerly Spark-CLR) is a cross-company open source project at Microsoft Research that aims to provide C# language bindings for Apache Spark. There is also a considerable body of research on distributed frameworks, including Apache Spark, for solving big data challenges [3,27].

10 Conclusions

In this paper, we have introduced a review on the key features of Apache Spark for big data analytics. Apache Spark is a general-purpose cluster computing framework with an optimized engine that supports advanced execution DAGs and APIs in Java, Scala, Python and R. Spark's MLlib, including the ML pipelines API, provides a variety of functionalities for designing, implementing and tuning machine learning algorithms and pipelines. GraphX is built on top of property graphs (i.e., an extension of RDDs for graph representation) and comes with a collection of operators to simplify graph analysis. Spark Streaming is built on top of DStreams (i.e., an extension of RDDs for stream data) and supports a wide range of operations and data sources.

---

[60] https://github.com/yahoo/CaffeOnSpark.

[61] http://tinyurl.com/zpn4qay.

[62] https://github.com/Microsoft/Mobius.

While RDD is the basic abstraction and the RDD API will remain the low-level API, two other alternatives are under active development now: the DataFrame API and the Dataset API. These alternatives are becoming the backbone of Apache Spark for better data representation and computation optimization. Current efforts in this regard include, but are not limited to GraphFrames, Structured Streaming, and the Tungsten project as a whole.

Considering the upper-level libraries which are built on top of Spark core, Apache Spark provides a unified engine which goes beyond batch processing to combine different workloads such as iterative algorithms, streaming and interactive queries. It is apparent that Apache Spark project, supported by other projects from academia and industry, has already done an essential contribution for solving key challenges of big data analytics. However, the big data community still needs more in-depth analyses of Apache Spark's performance in different scenarios, although there are several endeavors for Apache Spark's benchmarking.

References

1. Agarwal, S., Mozafari, B., Panda, A., Milner, H., Madden, S., Stoica, I.: Blinkdb: queries with bounded errors and bounded response times on very large data. In: Proceedings of the 8th ACM European Conference on Computer Systems. ACM, New York, pp 29–42 (2013). doi:10.1145/2465351.2465355
2. Amde, M., Bradley, J.: Scalable decision trees in mllib. https://databricks.com/blog/2014/09/29/scalable-decision-trees-in-mllib.html (2014)
3. Anagnostopoulos, I., Zeadally, S., Exposito, E.: Handling big data: research challenges and future directions. J. Supercomput. (2016). doi:10.1007/s11227-016-1677-z
4. Andrew, G., Gao, J.: Scalable training of l1-regularized log-linear models. In: International Conference on Machine Learning (2007)
5. Apiletti, D., Garza, P., Pulvirenti, F.: New Trends in databases and information systems: ADBIS 2015 Short Papers and Workshops, BigDap, DCSA, GID, MEBIS, OAIS, SW4CH, WISARD, Poitiers, France, September 8–11, 2015. Proceedings, Springer International Publishing, Cham, chap A Review of Scalable Approaches for Frequent Itemset Mining, pp. 243–247 (2015)
6. Aridhi, S., Nguifo, E.M.: Big graph mining: frameworks and techniques. arXiv preprint arXiv:1602.03072 (2016)
7. Armbrust, M., Ghodsi, A., Zaharia, M., Xin, R.S., Lian, C., Huai, Y., Liu, D., Bradley, J.K., Meng, X., Kaftan, T., Franklin, M.J.: Spark SQL. In: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data—SIGMOD '15, ACM Press, New York, NY, USA, pp. 1383–1394. doi:10.1145/2723372.2742797. http://dl.acm.org/citation.cfm?id=2723372.2742797 (2015)
8. Armbrust, M., Huai, Y., Liang, C., Xin, R., Zaharia, M.: Deep dive into spark sqls catalyst optimizer. https://databricks.com/blog/2015/04/13/deep-dive-into-spark-sqls-catalyst-optimizer.html (2015)
9. Armbrust, M., Fan, W., Xin, R., Zaharia, M.: Introducing spark datasets. https://databricks.com/blog/2016/01/04/introducing-spark-datasets.html (2016)

10. Awan, A.J., Brorsson, M., Vlassov, V., Ayguadé, E.: How data volume affects spark based data analytics on a scale-up server. CoRR arxiv:1507.08340 (2015)

11. Awan, A.J., Brorsson, M., Vlassov, V., Ayguadé, E.: Architectural impact on performance of in-memory data analytics: Apache spark case study. CoRR arxiv:1604.08484 (2016)

12. Boehm, M., Tatikonda, S., Reinwald, B., Sen, P., Tian, Y., Burdick, D.R., Vaithyanathan, S.: Hybrid parallelization strategies for large-scale machine learning in systemML. Proc. VLDB Endow. **7**(7), 553–564 (2014). doi:10.14778/2732286.2732292

13. Bu, Y., Howe, B., Balazinska, M., Ernst, M.D.: Haloop: efficient iterative data processing on large clusters. Proc. VLDB Endow. **3**(1–2), 285–296 (2010). doi:10.14778/1920841.1920881

14. Burdorf, C.: Use of spark mllib for predicting the offlining of digital media. Presentation. https://spark-summit.org/2015/events/use-of-spark-mllib-for-predicting-the-offlining-of-digital-media/ (2015)

15. Busa, N.: Real-time anomaly detection with spark ml and akka. Presentation. https://spark-summit.org/eu-2015/events/real-time-anomaly-detection-with-spark-ml-and-akka/ (2015)

16. Capotă, M, Hegeman, T., Iosup, A., Prat-Pérez, A., Erling, O., Boncz, P.: Graphalytics: a big data benchmark for graph-processing platforms. In: Proceedings of the GRADES'15, ACM, New York, NY, USA, GRADES'15, pp. 7:1–7:6. doi:10.1145/2764947.2764954 (2015)

17. Chakrabarti, D., Zhan, Y., Faloutsos, C.: R-mat: a recursive model for graph mining. In: In Fourth SIAM International Conference on Data Mining (2004)

18. Chan, W.: Databricks democratizes data and reduces infrastructure costs for eyeview. https://databricks.com/blog/2016/02/03/databricks-democratizes-data-and-reduces-infrastructure-costs-for-eyeview.html (2016)

19. Cheng, R., Chen, E., Hong, J., Kyrola, A., Miao, Y., Weng, X., Wu, M., Yang, F., Zhou, L., Zhao, F.: Kineograph. In: Proceedings of the 7th ACM european conference on Computer Systems—EuroSys '12, ACM Press, New York, NY, USA, p 85. doi:10.1145/2168836.2168846. http://dl.acm.org/citation.cfm?id=2168836.2168846 (2012)

20. Crankshaw, D., Bailis, P., Gonzalez, J.E., Li, H., Zhang, Z., Franklin, M.J., Ghodsi, A., Jordan, M.I.: The missing piece in complex analytics: low latency, scalable model management and serving with velox. CoRR arxiv:1409.3809 (2014)

21. Damji, J.: A tale of three apache spark apis: Rdds, dataframes, and datasets. https://databricks.com/blog/2016/07/14/a-tale-of-three-apache-spark-apis-rdds-dataframes-and-datasets.html (2016)

22. Das, T., Zaharia, M., Wendell, P.: Diving into spark streaming's execution model. https://databricks.com/blog/2015/07/30/diving-into-spark-streamings-execution-model.html (2015)

23. Databricks: Databricks spark reference applications. http://tinyurl.com/gwzkqxr (2015)

24. Dave, A.: Graphframes: graph queries in spark sql. Presentation. https://spark-summit.org/east-2016/events/graphframes-graph-queries-in-spark-sql/ (2016)

25. Dave, A., Jindal, A., Li, L.E., Xin, R., Gonzalez, J., Zaharia, M.: Graphframes: an integrated api for mixing graph and relational queries. In: Proceedings of the Fourth International Workshop on Graph Data Management Experiences and Systems, ACM, New York, NY, USA, GRADES '16, pp. 2:1–2:8. doi:10.1145/2960414.2960416 (2016)

26. Ekanayake, J., Li, H., Zhang, B., Gunarathne, T., Bae, S.H., Qiu, J., Fox, G.: Twister: A runtime for iterative mapreduce. In: Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, ACM, New York, NY, USA, HPDC '10, pp. 810–818. doi:10.1145/1851476.1851593 (2010)

27. Fernndez, A., del Ro, S., Lpez, V., Bawakid, A., del Jesus, M.J., Bentez, J.M., Herrera, F.: Big data with cloud computing: an insight on the computing environment, mapreduce, and programming frameworks. Wiley Interdiscip. Rev. Data Min. Knowl. Discov. **4**(5), 380–409 (2014). doi:10.1002/widm.1134

28. Freeman, J.: A platform for large-scale neuroscience. Presentation. https://spark-summit.org/2014/talk/A-platform-for-large-scale-neuroscience (2014)

29. Freeman, J.: Introducing streaming k-means in spark 1.2. https://databricks.com/blog/2015/01/28/introducing-streaming-k-means-in-spark-1-2.html (2015)

30. Freeman, J.: Open source tools for large-scale neuroscience. Curr. Opin. Neurobiol. **32**, 156–163 (2015). doi:10.1016/j.conb.2015.04.002. large-Scale Recording Technology (32)

31. Ganelin, l: Spark: Big Data Cluster Computing in Production. Wiley, New York (2016)

32. Ghoting, A., Krishnamurthy, R., Pednault, E.P.D., Reinwald, B., Sindhwani, V., Tatikonda, S., Tian, Y., Vaithyanathan, S.: Systemml: Declarative machine learning on mapreduce. In: Abiteboul, S., Böhm, K., Koch, C., Tan, K. (eds.) Proceedings of the 27th International Conference on Data Engineering, ICDE 2011, April 11–16, 2011, Hannover, Germany, IEEE Computer Society, pp. 231–242. doi:10.1109/ICDE.2011.5767930 (2011)

33. Gonzalez, J.E.: From graphs to tables the design of scalable systems for graph analytics. In: 23rd International World Wide Web Conference, WWW '14, Seoul, Republic of Korea, April 7–11, 2014, Companion Volume, pp. 1149–1150. doi:10.1145/2567948.2580059 (2014)

34. Gonzalez, J.E., Low, Y., Gu, H., Bickson, D., Guestrin, C.: PowerGraph: distributed graph-parallel computation on natural graphs, pp. 17–30. http://dl.acm.org/citation.cfm?id=2387880.2387883 (2012)

35. Gonzalez, J.E., Xin, R.S., Dave, A., Crankshaw, D., Franklin, M.J., Stoica, I.: Graphx: Graph processing in a distributed dataflow framework. In: Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation, USENIX Association, Berkeley, CA, USA, OSDI'14, pp. 599–613. http://dl.acm.org/citation.cfm?id=2685048.2685096 (2014)

36. Gopalani, S., Arora, R.: Article: Comparing apache spark and map reduce with performance analysis using k-means. Int. J. Comput. Appl. **113**(1), 8–11 (2015). (full text available)

37. Guller, M.: Big Data Analytics with Spark: A Practitioner's Guide to Using Spark for Large Scale Data Analysis. Apress. https://books.google.de/books?id=bNP8rQEACAAJ (2015)

38. Gulzar, M.A., Interlandi, M., Yoo, S., Tetali, S.D., Condie, T., Millstein, T., Kim, M.: Bigdebug: debugging primitives for interactive big data processing in spark. In: Proceedings of 38th IEEE/ACM International Conference on Software Engineering, ICSE' 16 (2016)

39. Hindman, B., Konwinski, A., Zaharia, M., Ghodsi, A., Joseph, A.D., Katz, R., Shenker, S., Stoica, I.: Mesos: a platform for fine-grained resource sharing in the data center. In: Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation, USENIX Association, Berkeley, CA, USA, NSDI'11, pp. 295–308. http://dl.acm.org/citation.cfm?id=1972457.1972488 (2011)

40. Huang, M.: Dynamic community detection for large-scale e-commerce data with spark streaming and graphx. Presentation. https://spark-summit.org/2015/events/hybrid-community-detection-for-web-scale-e-commerce-using-spark-streaming-and-graphx/ (2015)

41. Interlandi, M., Shah, K., Tetali, S.D., Gulzar, M., Yoo, S., Kim, M., Millstein, T.D., Condie, T.: Titian: Data provenance support in spark. PVLDB **9**(3), 216–227. http://www.vldb.org/pvldb/vol9/p216-interlandi.pdf (2015)

42. Ivanov, T., Beer, M.: Evaluating Hive and spark SQL with bigbench. CoRR arxiv:1512.08417 (2015)

43. Iyer, A.P., Li, L.E., Das, T., Stoica, I.: Time-evolving graph processing at scale. In: Proceedings of the Fourth International Workshop on Graph Data Management Experiences and Systems, ACM, New York, NY, USA, GRADES '16, pp. 5:1–5:6. doi:10.1145/2960414.2960419 (2016)

44. Jarrah, M., Al-Quraan, M., Jararweh, Y., Al-Ayyoub, M.: Medgraph: a graph-based representation and computation to handle large sets of images. Multimedia Tools and Applications, pp. 1–17. doi:10.1007/s11042-016-3262-0 (2016)

45. Karau, H., Konwinski, A., Wendell, P., Zaharia, M.: Learning Spark: Lightning-Fast Big Data Analytics, 1st edn. O'Reilly Media, Inc, Sebastopol (2015)

46. Kim, H., Park, J., Jang, J., Yoon, S.: Deepspark: Spark-based deep learning supporting asynchronous updates and caffe compatibility. CoRR arxiv:1602.08191 (2016)

47. Koren, Y.: Factorization meets the neighborhood: a multifaceted collaborative filtering model. In: Li, Y., Liu, B., Sarawagi, S. (eds.) Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Las Vegas, Nevada, USA, August 24–27, 2008, ACM, pp. 426–434. doi:10.1145/1401890.1401944 (2008)

48. Kraska, T., Talwalkar, A., Duchi, J.C., Griffith, R., Franklin, M.J., Jordan, M.I.: Mlbase: A distributed machine-learning system. In: CIDR. www.cidrdb.org. http://dblp.uni-trier.de/db/conf/cidr/cidr2013.html (2013)

49. Krishnan, D.R., Quoc, D.L., Bhatotia, P., Fetzer, C., Rodrigues, R.: Incapprox: A data analytics system for incremental approximate computing. In: Proceedings of the 25th International Conference on World Wide Web, International World Wide Web Conferences Steering Committee, pp. 1133–1144 (2016)

50. Kursar, B.: Data driven—toyota customer 360 insights on apache spark and mllib. Presentation. https://spark-summit.org/2015/events/keynote-7/ (2015)

51. Landset, S., Khoshgoftaar, T.M., Richter, A.N., Hasanin, T.: A survey of open source tools for machine learning with big data in the hadoop ecosystem. J. Big Data 2(1), 1–36 (2015). doi:10.1186/s40537-015-0032-1

52. Li, H., Ghodsi, A., Zaharia, M., Shenker, S., Stoica, I.: Tachyon: Reliable, memory speed storage for cluster computing frameworks. In: Proceedings of the ACM Symposium on Cloud Computing, ACM, pp. 1–15 (2014)

53. Li, M., Tan, J., Wang, Y., Zhang, L., Salapura, V.: SparkBench. In: Proceedings of the 12th ACM International Conference on Computing Frontiers—CF '15, ACM Press, New York, New York, USA, pp. 1–8. doi:10.1145/2742854.2747283 (2015)

54. Li, P., Luo, Y., Zhang, N., Cao, Y.: Heterospark: A heterogeneous cpu/gpu spark platform for machine learning algorithms. In: 2015 IEEE International Conference on Networking, Architecture and Storage (NAS), pp. 347–348. doi:10.1109/NAS.2015.7255222 (2015)

55. Low, Y., Gonzalez, J., Kyrola, A., Bickson, D., Guestrin, C., Hellerstein, J.M.: GraphLab: A New Framework for Parallel Machine Learning, pp. 8–11. arxiv:1006.4990 (2010)

56. Low, Y., Bickson, D., Gonzalez, J., Guestrin, C., Kyrola, A., Hellerstein, J.M.: Distributed graphlab: a framework for machine learning and data mining in the cloud. Proc. VLDB Endow. 5(8), 716–727 (2012). doi:10.14778/2212351.2212354

57. Malewicz, G., Austern, M.H., Bik, A.J., Dehnert, J.C., Horn, I., Leiser, N., Czajkowski, G.: Pregel. In: Proceedings of the 2010 International Conference on Management of data—SIGMOD '10, ACM Press, New York, NY, USA, p 135. http://dl.acm.org/citation.cfm?id=1807167.1807184 (2010)

58. Malewicz, G., Austern, M.H., Bik, A.J., Dehnert, J.C., Horn, I., Leiser, N., Czajkowski, G.: Pregel: A system for large-scale graph processing. In: Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data, ACM, New York, NY, USA, SIGMOD '10, pp. 135–146. doi:10.1145/1807167.1807184 (2010)

59. Marcu, O.C., Costan, A., Antoniu, G., Pérez, M.S.: Spark versus Flink: Understanding Performance in Big Data Analytics Frameworks. In: Cluster 2016—The IEEE 2016 International Conference on Cluster Computing, Taipei, Taiwan. https://hal.inria.fr/hal-01347638 (2016)

60. Massie, M., Nothaft, F., Hartl, C., Kozanitis, C., Schumacher, A., Joseph, A.D., Patterson, D.A.: Adam: Genomics formats and processing patterns for butt scale computing. Tech. Rep. UCB/EECS-2013-207, EECS Department, University of California, Berkeley (2013)

61. Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., Freeman, J., Tsai, D., Amde, M., Owen, S., Xin, D., Xin, R., Franklin, M.J., Zadeh, R., Zaharia, M., Talwalkar, A.: Mllib: Machine learning in apache spark arxiv:1505.06807 (2015)

62. Moffitt, V.Z., Stoyanovich, J.: Portal: a query language for evolving graphs. arXiv preprint arXiv:1602.00773 (2016)

63. Moffitt, V.Z., Stoyanovich, J.: Towards a distributed infrastructure for evolving graph analytics. https://www.cs.drexel.edu/~julia/documents/tempweb16.pdf (2016)

64. Moritz, P., Nishihara, R., Stoica, I., Jordan, M.I.: Sparknet: Training deep networks in spark. CoRR arxiv:1511.06051 (2015)

65. O'Brien, A.R., Saunders, N.F.W., Guo, Y., Buske, F.A., Scott, R.J., Bauer, D.C.: Variantspark: population scale clustering of genotype information. BMC Genom. 16(1), 1–9 (2015). doi:10.1186/s12864-015-2269-7

66. Ousterhout, K., Rasti, R., Ratnasamy, S., Shenker, S., Chun, B.G.: Making sense of performance in data analytics frameworks. In: Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation, USENIX Association, Berkeley, CA, USA, NSDI'15, pp. 293–307. http://dl.acm.org/citation.cfm?id=2789770.2789791 (2015)

67. Palamuttam, R., Mogrovejo, R.M., Mattmann, C., Wilson, B., Whitehall, K., Verma, R., McGibbney, L.J., Ramirez, P.M.: Scispark: applying in-memory distributed computing to weather event detection and tracking. In: 2015 IEEE International Conference on Big Data, Big Data 2015, Santa Clara, CA, USA, October 29-November 1, 2015, IEEE, pp. 2020–2026. doi:10.1109/BigData.2015.7363983 (2015)

68. Ramrez-Gallego, S., Garca, S., Mourio-Taln, H., Martnez-Rego, D., Boln-Canedo, V., Alonso-Betanzos, A., Bentez, J.M., Herrera, F.: Data discretization: taxonomy and big data challenge. Wiley Interdiscip. Rev. Data Min. Knowl. Discov. 6(1), 5–21 (2016). doi:10.1002/widm.1173

69. Richter, A.N., Khoshgoftaar, T.M., Landset, S., Hasanin, T.: A multi-dimensional comparison of toolkits for machine learning with big data. In: 2015 IEEE International Conference on Information Reuse and Integration, IRI 2015, San Francisco, CA, USA, August 13–15, 2015, IEEE, pp. 1–8. doi:10.1109/IRI.2015.12 (2015)

70. Ryza, S., Laserson, U., Owen, S., Wills, J.: Advanced Analytics with Spark: Patterns for Learning from Data at Scale. O'Reilly Media. https://books.google.de/books?id=M0_GBwAAQBAJ (2015)

71. Salperwyck, C., Maby, S., Cubillé, J., Lagacherie, M.: Courbospark: Decision tree for time-series on spark. In: Proceedings of the 1st International Workshop on Advanced Analytics and Learning on Temporal Data, AALTD 2015, co-located with The European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD 2015), Porto, Portugal, September 11, 2015. http://ceur-ws.org/Vol-1425/paper15.pdf (2015)

72. Shi, J., Qiu, Y., Minhas, U.F., Jiao, L., Wang, C., Reinwald, B., Özcan, F.: Clash of the titans: Mapreduce vs. spark for large scale data analytics. Proc. VLDB Endow. **8**(13), 2110–2121 (2015). doi:10.14778/2831360.2831365

73. Shyam, R., Kumar, S., Poornachandran, P., Soman, K.P.: Apache spark a big data analytics platform for smart grid. Proc. Technol. **21**, 171–178 (2015). doi:10.1016/j.protcy.2015.10.085

74. Sparks, E.R., Talwalkar, A., Franklin, M.J., Jordan, M.I., Kraska, T.: Tupaq: An efficient planner for large-scale predictive analytic queries. CoRR arxiv:1502.00068 (2015)

75. Sparks, E.R., Talwalkar, A., Haas, D., Franklin, M.J., Jordan, M.I., Kraska, T.: Automating model search for large scale machine learning. In: Proceedings of the Sixth ACM Symposium on Cloud Computing, ACM, New York, NY, USA, SoCC '15, pp. 368–380. doi:10.1145/2806777.2806945 (2015)

76. Vavilapalli, V.K., Murthy, A.C., Douglas, C., Agarwal, S., Konar, M., Evans, R., Graves, T., Lowe, J., Shah, H., Seth, S., Saha, B., Curino, C., O'Malley, O., Radia, S., Reed, B., Baldeschwieler, E.: Apache hadoop yarn: yet another resource negotiator. In: Proceedings of the 4th Annual Symposium on Cloud Computing, ACM, New York, NY, USA, SOCC '13, pp. 5:1–5:16. doi:10.1145/2523616.2523633 (2013)

77. Venkataraman, S., Yang, Z., Liu, D., Liang, E., Falaki, H., Meng, X., Xin, R., Ghodsi, A., Franklin, M., Stoica, I., Zaharia, M.: Sparkr: Scaling r programs with spark. In: Proceedings of the 2016 International Conference on Management of Data, ACM, New York, NY, USA, SIGMOD '16, pp. 1099–1104. doi:10.1145/2882903.2903740 (2016)

78. Wang, K., Khan, M.M.H.: Performance prediction for apache spark platform. In: 2015 IEEE 17th International Conference on High Performance Computing and Communications (HPCC), 2015 IEEE 7th International Symposium on Cyberspace Safety and Security (CSS), 2015 IEEE 12th International Conferen on Embedded Software and Systems (ICESS), pp. 166–173. doi:10.1109/HPCC-CSS-ICESS.2015.246 (2015)

79. Xiao, B.: Huawei embraces open-source apache spark. https://databricks.com/blog/2015/06/09/huawei-embraces-open-source-apache-spark.html (2015)

80. Xin, R.: Spark officially sets a new record in large-scale sorting. https://databricks.com/blog/2014/11/05/spark-officially-sets-a-new-record-in-large-scale-sorting.html (2014)

81. Xin, R.: Technical preview of apache spark 2.0 now on databricks. https://databricks.com/blog/2016/05/11/apache-spark-2-0-technical-preview-easier-faster-and-smarter.html (2016)

82. Xin, R., Rosen, J.: Project tungsten: Bringing spark closer to bare metal. Presentation. https://databricks.com/blog/2015/04/28/project-tungsten-bringing-spark-closer-to-bare-metal.html (2015)

83. Xin, R.S., Gonzalez, J.E., Franklin, M.J., Stoica, I.: Graphx: a resilient distributed graph system on spark. In: First International Workshop on Graph Data Management Experiences and Systems, GRADES 2013, co-loated with SIGMOD/PODS 2013, New York, NY, USA, June 24, 2013, p 2. http://event.cwi.nl/grades2013/02-xin.pdf (2013)

84. Xin, R.S., Rosen, J., Zaharia, M., Franklin, M.J., Shenker, S., Stoica, I.: Shark: Sql and rich analytics at scale. In: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, ACM, New York, NY, USA, SIGMOD '13, pp. 13–24. doi:10.1145/2463676.2465288 (2013)

85. Xin, R.S., Crankshaw, D., Dave, A., Gonzalez, J.E., Franklin, M.J., Stoica, I.: Graphx: Unifying data-parallel and graph-parallel analytics. CoRR arxiv:1402.2394 (2014)

86. Yan, D., Cheng, J., Ozsu, M.T., Yang, F., Lu, Y., Lui, J.C.S., Zhang, Q., Ng,W.: A general-purpose query-centric framework for querying big graphs. Proc. VLDB Endow. **9**(7), 564–575 (2016). doi:10.14778/2904483.2904488

87. Yu, J., Jinxuan, W., Mohamed, S.: GeoSpark: A Cluster Computing Framework for Processing Large-Scale Spatial Data. In: 23th International Conference on Advances in Geographic Information Systems. http://www.public.asu.edu/~jinxuanw/papers/GeoSpark.pdf (2015)

88. Yu, Y., Isard, M., Fetterly, D., Budiu, M., Erlingsson, U., Gunda, P.K., Currey, J.: Dryadlinq: A system for general-purpose distributed data-parallel computing using a high-level language. In: Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation, USENIX Association, Berkeley, CA, USA, OSDI'08, pp. 1–14. http://dl.acm.org/citation.cfm?id=1855741.1855742 (2008)

89. Zadeh, R.B., Meng, X., Yavuz, B., Staple, A., Pu, L., Venkataraman, S., Sparks, E., Ulanov, A., Zaharia, M.: linalg: Matrix computations in apache spark. arxiv:1509.02256 (2015)

90. Zaharia, M.: An Architecture for Fast and General Data Processing on Large Clusters. Association for Computing Machinery, New York, NY, USA (2016)

91. Zaharia, M.: Spark 2.0. Presentation. http://www.slideshare.net/databricks/2016-spark-summit-east-keynote-matei-zaharia (2016)

92. Zaharia, M., Wendell, P.: Spark community update. Presentation. http://www.slideshare.net/databricks/spark-community-update-spark-summit-san-francisco-2015 (2015)

93. Zaharia, M., Chowdhury, M., Franklin, M.J., Shenker, S., Stoica, I.: Spark: cluster computing with working sets p 10. http://dl.acm.org/citation.cfm?id=1863103.1863113 (2010)

94. Zaharia, M., Chowdhury, M., Das, T., Dave, A.: Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. NSDI'12 Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation pp. 2–2. doi:10.1111/j.1095-8649.2005.00662.x (2012)

95. Zaharia, M., Das, T., Li, H., Hunter, T., Shenker, S., Stoica, I.: Discretized streams: fault-tolerant streaming computation at scale. In: Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles, ACM, New York, NY, USA, SOSP '13, pp. 423–438. doi:10.1145/2517349.2522737 (2013)

96. Zhang, Y., Jordan, M.I.: Splash: User-friendly programming interface for parallelizing stochastic algorithms. CoRR arxiv:1506.07552 (2015)

97. Zhao, G., Ling, C., Sun, D.: Sparksw: Scalable distributed computing system for large-scale biological sequence alignment. In: 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid 2015, Shenzhen, China, May 4–7, 2015, IEEE Computer Society, pp. 845–852. doi:10.1109/CCGrid.2015.55 (2015)

98. Zhu, B., Mara, A., Mozo, A.: New Trends in Databases and Information Systems: ADBIS 2015 Short Papers and Workshops, BigDap, DCSA, GID, MEBIS, OAIS, SW4CH, WISARD, Poitiers, France, September 8–11, 2015. Proceedings, Springer International Publishing, Cham, chap CLUS: Parallel Subspace Clustering Algorithm on Spark, pp. 175–185 (2015)