What is Apache Spark? The big data platform that crushed Hadoop

Fast, flexible, and developer-friendly, Apache Spark is the leading platform for large-scale SQL, batch processing, stream processing, and machine learning

By Ian Pointer

InfoWorld |
MAR 16, 2020 3:00 AM PDT

Apache Spark defined

Apache Spark is a data processing framework that can quickly perform processing tasks on very large data sets, and can also distribute data processing tasks across multiple computers, either on its own or in tandem with other distributed computing tools. These two qualities are key to the worlds of big data and machine learning, which require the marshalling of massive computing power to crunch through large data stores. Spark also takes some of the programming burdens of these tasks off the shoulders of developers with an easy-to-use API that abstracts away much of the grunt work of distributed computing and big data processing.

From its humble beginnings in the AMPLab at U.C. Berkeley in 2009, Apache Spark has become one of the key big data distributed processing frameworks in the world. Spark can be deployed in a variety of ways, provides native bindings for the Java, Scala, Python, and R programming languages, and supports SQL, streaming data, machine learning, and graph processing. You'll find it used by banks, telecommunications companies, games companies, governments, and all of the major tech giants such as Apple, Facebook, IBM, and Microsoft.

[InfoWorld's 2020 Technology of the Year Award winners: The best software development, cloud computing, data analytics, and machine learning products of the year]

Apache Spark architecture

At a fundamental level, an Apache Spark application consists of two main components: a *driver,* which converts the user's code into multiple tasks that can be distributed across worker nodes, and *executors,* which run on those nodes and execute the tasks assigned to them. Some form of cluster manager is necessary to mediate between the two.

Out of the box, Spark can run in a standalone cluster mode that simply requires the Apache Spark framework and a JVM on each machine in your cluster. However, it's more likely you'll want to take advantage of a more robust resource or cluster management system to take care of allocating workers on demand for you. In the enterprise, this will normally mean running on Hadoop YARN (this is how the Cloudera and Hortonworks distributions run Spark jobs), but Apache Spark can also run on Apache Mesos, Kubernetes, and Docker Swarm.

If you seek a managed solution, then Apache Spark can be found as part of Amazon EMR, Google Cloud Dataproc, and Microsoft Azure HDInsight. Databricks, the company that employs the founders of Apache Spark, also offers the Databricks Unified Analytics Platform, which is a comprehensive managed service that offers Apache Spark clusters, streaming support, integrated web-based notebook development, and optimized cloud I/O performance over a standard Apache Spark distribution.

Apache Spark builds the user's data processing commands into a *Directed Acyclic Graph*, or DAG. The DAG is Apache Spark's scheduling layer; it determines what tasks are executed on what nodes and in what sequence.

Spark vs. Hadoop: Why use Apache Spark?

It's worth pointing out that Apache Spark vs. Apache Hadoop is a bit of a misnomer. You'll find Spark included in most Hadoop distributions these days. But due to two big advantages, Spark has become the framework of choice when processing big data, overtaking the old MapReduce paradigm that brought Hadoop to prominence.

The first advantage is speed. Spark's in-memory data engine means that it can perform tasks up to one hundred times faster than MapReduce in certain situations, particularly when compared with multi-stage jobs that require the writing of state back out to disk between stages. In essence, MapReduce creates a two-stage execution graph consisting of data mapping and reducing, whereas Apache Spark's DAG has multiple stages that can be distributed more efficiently. Even Apache Spark jobs where the data cannot be completely contained within memory tend to be around 10 times faster than their MapReduce counterpart.

The second advantage is the developer-friendly Spark API. As important as Spark's speedup is, one could argue that the friendliness of the Spark API is even more important.

Spark Core

In comparison to MapReduce and other Apache Hadoop components, the Apache Spark API is very friendly to developers, hiding much of the complexity of a distributed processing engine behind simple method calls. The canonical example of this is how almost 50 lines of MapReduce code to count words in a document can be reduced to just a few lines of Apache Spark (here shown in Scala):

```
val textFile = sparkSession.sparkContext.textFile(“hdfs:///tmp/words”)
val counts = textFile.flatMap(line => line.split(“ “))
                     .map(word => (word, 1))
                     .reduceByKey(_ + _)
counts.saveAsTextFile(“hdfs:///tmp/words_agg”)
```

By providing bindings to popular languages for data analysis like Python and R, as well as the more enterprise-friendly Java and Scala, Apache Spark allows everybody from application developers to data scientists to harness its scalability and speed in an accessible manner.

[Also on InfoWorld: Deep learning vs. machine learning: Understand the differences]

Spark RDD

At the heart of Apache Spark is the concept of the Resilient Distributed Dataset (RDD), a programming abstraction that represents an immutable collection of objects that can be split across a computing cluster. Operations on the RDDs can also be split across the cluster and executed in a parallel batch process, leading to fast and scalable parallel processing.

RDDs can be created from simple text files, SQL databases, NoSQL stores (such as Cassandra and MongoDB), Amazon S3 buckets, and much more besides. Much of the Spark Core API is built on this RDD concept, enabling traditional map and reduce functionality, but also providing built-in support for joining data sets, filtering, sampling, and aggregation.

Spark runs in a distributed fashion by combining a *driver* core process that splits a Spark application into tasks and distributes them among many *executor* processes that do the work. These executors can be scaled up and down as required for the application’s needs.

Spark SQL

Originally known as Shark, Spark SQL has become more and more important to the Apache Spark project. It is likely the interface most commonly used by today’s developers when creating applications. Spark SQL is focused on the processing of structured data, using a dataframe approach borrowed from R and Python (in Pandas). But as the name suggests, Spark SQL also provides a SQL2003-compliant interface for querying data, bringing the power of Apache Spark to analysts as well as developers.

Alongside standard SQL support, Spark SQL provides a standard interface for reading from and writing to other datastores including JSON, HDFS, Apache Hive, JDBC, Apache ORC, and Apache Parquet, all of which are supported out of the box. Other popular stores—Apache Cassandra, MongoDB, Apache HBase, and many others—can be used by pulling in separate connectors from the Spark Packages ecosystem.

Selecting some columns from a dataframe is as simple as this line:

```
citiesDF.select(“name”, “pop”)
```

Using the SQL interface, we register the dataframe as a temporary table, after which we can issue SQL queries against it:

```
citiesDF.createOrReplaceTempView(“cities”)
spark.sql(“SELECT name, pop FROM cities”)
```

Behind the scenes, Apache Spark uses a query optimizer called Catalyst that examines data and queries in order to produce an efficient query plan for data locality and computation that will perform the required calculations across the cluster. In the Apache Spark 2.x era, the Spark SQL interface of dataframes and datasets (essentially a typed dataframe that can be checked at compile time for correctness and take advantage of further memory and compute optimizations at run time) is the recommended approach for development. The RDD interface is still available, but recommended only if your needs cannot be addressed within the Spark SQL paradigm.

Spark 2.4 introduced a set of built-in higher-order functions for manipulating arrays and other higher-order data types directly.

Spark MLlib

Apache Spark also bundles libraries for applying machine learning and graph analysis techniques to data at scale. Spark MLlib includes a framework for creating machine learning pipelines, allowing for easy implementation of feature extraction, selections, and transformations on any structured dataset. MLlib comes with distributed implementations of clustering and classification algorithms such as k-means clustering and random forests that can be swapped in and out of custom pipelines with ease. Models can be trained by data scientists in Apache Spark using R or Python, saved using MLlib, and then imported into a Java-based or Scala-based pipeline for production use.

Note that while Spark MLlib covers basic machine learning including classification, regression, clustering, and filtering, it does not include facilities for modeling and training deep neural networks (for details see InfoWorld’s Spark MLlib review). However, Deep Learning Pipelines are in the works.

[Also on InfoWorld: Artificial intelligence predictions for 2020]

Spark GraphX

Spark GraphX comes with a selection of distributed algorithms for processing graph structures including an implementation of Google’s PageRank. These algorithms use Spark Core’s RDD approach to modeling data; the GraphFrames package allows you to do graph operations on dataframes, including taking advantage of the Catalyst optimizer for graph queries.

Spark Streaming

Spark Streaming was an early addition to Apache Spark that helped it gain traction in environments that required real-time or near real-time processing. Previously, batch and stream processing in the world of Apache Hadoop were separate things. You would write MapReduce code for your batch processing needs and use something like

Apache Storm for your real-time streaming requirements. This obviously leads to disparate codebases that need to be kept in sync for the application domain despite being based on completely different frameworks, requiring different resources, and involving different operational concerns for running them.

Spark Streaming extended the Apache Spark concept of batch processing into streaming by breaking the stream down into a continuous series of microbatches, which could then be manipulated using the Apache Spark API. In this way, code in batch and streaming operations can share (mostly) the same code, running on the same framework, thus reducing both developer and operator overhead. Everybody wins.

A criticism of the Spark Streaming approach is that microbatching, in scenarios where a low-latency response to incoming data is required, may not be able to match the performance of other streaming-capable frameworks like Apache Storm, Apache Flink, and Apache Apex, all of which use a pure streaming method rather than microbatches.

Structured Streaming

Structured Streaming (added in Spark 2.x) is to Spark Streaming what Spark SQL was to the Spark Core APIs: A higher-level API and easier abstraction for writing applications. In the case of Structure Streaming, the higher-level API essentially allows developers to create infinite streaming dataframes and datasets. It also solves some very real pain points that users have struggled with in the earlier framework, especially concerning dealing with event-time aggregations and late delivery of messages. All queries on structured streams go through the Catalyst query optimizer, and can even be run in an interactive manner, allowing users to perform SQL queries against live streaming data.

Structured Streaming originally relied on Spark Streaming's microbatching scheme of handling streaming data. But in Spark 2.3, the Apache Spark team added a low-latency Continuous Processing Mode to Structured Streaming, allowing it to handle responses with latencies as low as 1ms, which is very impressive. As of Spark 2.4, Continuous Processing is still considered experimental. While Structured Streaming is built on top of the Spark SQL engine, Continuous Streaming supports only a restricted set of queries.

Structured Streaming is the future of streaming applications with the platform, so if you're building a new streaming application, you should use Structured Streaming. The legacy Spark Streaming APIs will continue to be supported, but the project recommends porting over to Structured Streaming, as the new method makes writing and maintaining streaming code a lot more bearable.

Deep Learning Pipelines

Apache Spark supports deep learning via Deep Learning Pipelines. Using the existing pipeline structure of MLlib, you can call into lower-level deep learning libraries and construct classifiers in just a few lines of code, as well as apply custom TensorFlow graphs or Keras models to incoming data. These graphs and models can even be registered as custom Spark SQL UDFs (user-defined functions) so that the deep learning models can be applied to data as part of SQL statements.

Apache Spark tutorials

Ready to dive in and learn Apache Spark? We highly recommend Evan Heitman's A Neanderthal's Guide to Apache Spark in Python, which not only lays out the basics of how Apache Spark works in relatively simple terms, but also guides you through the process of writing a simple Python application that makes use of the framework.

The article is written from a data scientist's perspective, which makes sense as data science is a world in which big data and machine learning are increasingly critical.

[Keep up with advances in machine learning, AI, and big data analytics with InfoWorld's Machine Learning and Analytics Report newsletter]

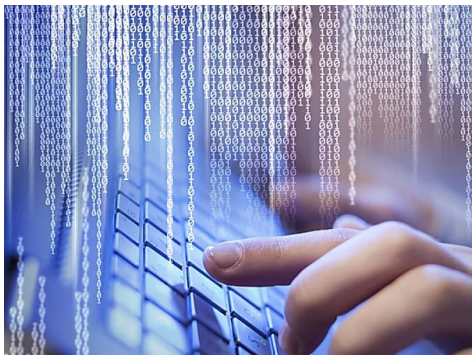
If you're looking for some Apache Spark examples to give you a sense of what the platform can do and how it does it, check out Spark By {Examples}. There is plenty of sample code here for a number of the basic tasks that make up the building blocks of Spark programming, so you can see the components that make up the larger tasks that Apache Spark is made for.

Need to go deeper? DZone has what it modestly refers to as The Complete Apache Spark Collection, which consists of a slew of helpful tutorials on many Apache Spark topics. Happy learning!

Ian Pointer is a senior big data and deep learning architect, working with Apache Spark and PyTorch. He has more than 15 years of development and operations experience.

Follow

9 ways to build privacy into your cloud applications

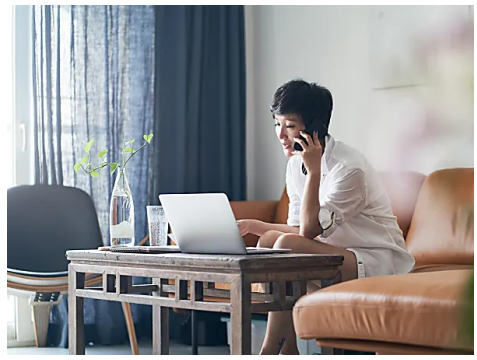


What's new in Microsoft Visual Studio Code 1.48

What's new in Microsoft .NET 5

Black developers tell how the US tech industry could do

2 cloud architecture problems are still unsolved



Using robotics and immersive technologies to support…

Social engineering hacks weaken cybersecurity during the pandemic

Why developers should use graph databases

UNITED STATES

/