What‘s the big deal with Spark anyway?

medium.com/it-s-a-data-world/what-s-the-big-deal-with-spark-anyway-f3c108924a08

If you live or work anywhere near Big Data these days you’ve heard about Spark, or more specifically Spark vs Hadoop. **As a marketer working for a “Big Data” company I had to figure out what the hype was all about.** I started my exploration by figuring out what Hadoop and distributed storage was all about. Now let’s try and figure out where Spark comes along…

So before we go any further, let’s be rigorous

Comparing Spark and Hadoop makes no sense.

I know, your whole belief system just collapsed.

Spark is an alternative to MapReduce. It doesn’t even have its own distributed storage system so **it’s generally installed on top of the Hadoop Distributed File System.** It can work directly on top of YARN or other cluster managers like MESOS. **So it works WITH Hadoop, instead of MapReduce**.

The main difference between Spark and MapReduce is that **MapReduce works with persistence storage, while Spark uses Resilient Distributed Datasets**. Yeah I know, uhm what?

Spark vs MapReduce

Basically when you do a query in **MapReduce, the system rewrites all the data back on the physical storage medium after each operation.** This makes it slow. It also means you can’t reuse intermediate results when you’re performing multiple computations. If you have an algorithm that performs the same operation repeatedly it can be very costly in terms of computation time.

Spark handles a lot of the operations “in memory”. So it copies the data from the physical storage medium into the RAM memory of the worker node, and processes it far faster. While it’s stored in RAM, Spark can also do several operations on the data at once.

There is a reason that MapReduce doesn't process operations in-memory of course and is therefore much slower. Originally when the data was processed in-memory, it was vulnerable to faults. The memory storage is much less stable that the physical medium. However, **Spark has managed to make in-memory processing fault tolerant with Resilient Distributed Datasets.** More generally, it means that if the RAM memory fails, the data can be recovered. If you really want to know how, (no, I can't tell you everything).

More generally, Spark doesn't read and write data as often so it can be between 10 to 100 times faster to execute jobs than MapReduce.

So now when someone talks to you about Spark vs Hadoop, you should first think:

- **Uhm it's Spark vs MapReduce, not Hadoop**
- **Spark is faster because it processes in-memory.**

So fast

There are a couple of other differencing elements.

Streaming data with Spark Streaming

Because Spark processes in-memory, **it isn't limited to functioning in batch mode** like MapReduce is. Batch processing means that when it processes data, it takes sets of records as an input, processes them, and then produces an output of sets of data. Both the input and the output are batches of records.

The opposite is **streaming processing; where you can stream data and analyze it live.** Your calculations can run on all your data as it comes in and have a truly iterative model. This is allowed by Spark Streaming. If you want to sound particularly cool at Machine Learning Meetups at this point you can point out that Spark Streaming isn't actually "streaming". In reality it works on micro batches not on each record at a time. But the effect is the same.

Spark for developers

Also MapReduce is hard to set-up and to program. **Spark makes it easier to program applications in familiar programming languages like Python or Java. Its API is also much more flexible than MapReduce's so it allows more versatility.** Moreover it can be set on all kinds of different data platforms, and can **access lots of different data sources** including HDFS, but also Cassandra or Amazon S3 (cloud storage).

Bye bye Hadoop

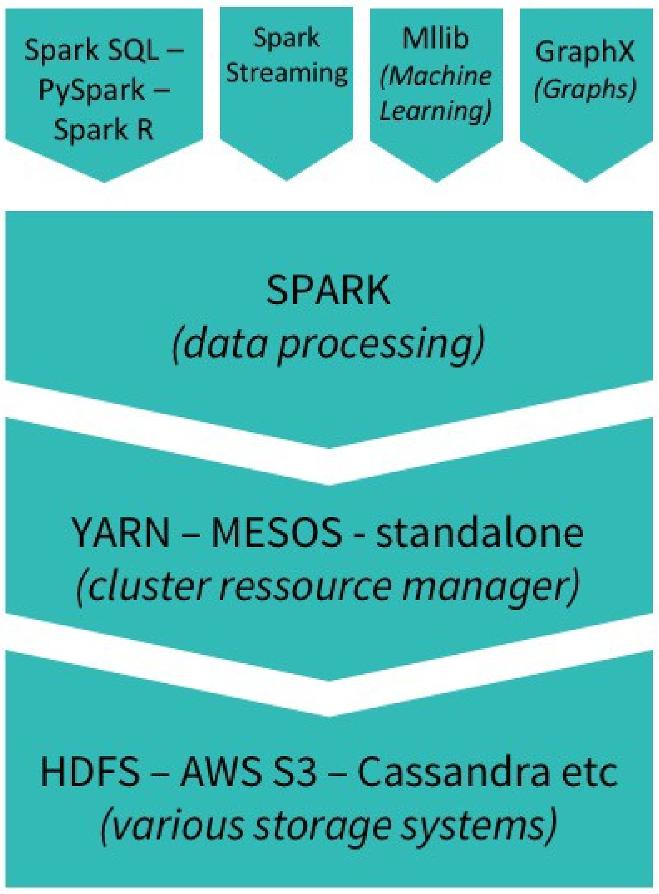
Another interesting aspect for developers is that Spark can come, as we've said, with Spark Streaming as well as a batch mode like Hadoop. This allows for real-time processing. So you can get insights from your data as it is captured and get faster business decisions.

Spark and Machine Learning

Spark is also particularly useful for Machine Learning, because you can reuse intermediate results of your computations. Many Machine Learning algorithms are based on several **iterations over the same data** to refine the results. This is much faster when you can work on intermediate results in RAM. MapReduce rewrites all intermediate results so the processing time is multiplied.

With distributed storage systems you also have a set-up time for jobs you compute. Because MapReduce rewrites every intermediate result for every new job you'll have to set it up again. Spark however will prepare the worker nodes to distribute several operations simultaneously, **so you only have to set-up once.** This means **it can implement complex algorithms in a more efficient manner without having to set-up at each new step.**

Spark also has a Machine Learning library developed for it which is one of the most dynamic open source projects: **MLLib. This algorithm library is only constituted of highly scalable algorithms that can be trained across distributed data.** You can also use Spark and MLLib in Python with PySpark. You can query your data in SQL with SparkSQL or process it in R with Spark R. And you can use GraphX to build graphs of your distributed data with Spark speed!


Spark SQL – PySpark – Spark R
Spark Streaming
Mllib (Machine Learning)
GraphX (Graphs)
SPARK (data processing)
YARN – MESOS - standalone (cluster ressource manager)
HDFS – AWS S3 – Cassandra etc (various storage systems)

The Spark stack

Since **I have now said Spark 36 times in this article**, I think it’s time to take a break and ponder on how great Spark is. Before we can start thinking about why it probably isn’t the ultimate solution to all your data issues. TTYL