**Introduction**

**Why Learn Spark?**

Spark is currently one of the most popular tools for big data analytics. You might have heard of other tools such as Hadoop. Hadoop is a slightly older technology although still in use by some companies. Spark is generally faster than Hadoop, which is why Spark has become more popular over the last few years.

There are many other big data tools and systems, each with its own use case. For example, there are database system like [Apache Cassandra](http://cassandra.apache.org/) and SQL query engines like [Presto](https://prestodb.io/). But Spark is still one of the most popular tools for analyzing large data sets.

Here is an outline of the topics we are covering in this lesson:

* What is big data?
* Review of the hardware behind big data
* Introduction to distributed systems
* Brief history of Spark and big data
* Common Spark use cases
* Other technologies in the big data ecosystem

In the next few videos, you'll learn about four key hardware components. Understanding these components helps determine whether you are working on a "big data" problem or if it's easier to analyze the data locally on your own computer.

**CPU (Central Processing Unit)**  
The CPU is the "brain" of the computer. Every process on your computer is eventually handled by your CPU. This includes calculations and also instructions for the other components of the compute.

**Memory (RAM)**  
When your program runs, data gets temporarily stored in memory before getting sent to the CPU. Memory is ephemeral storage - when your computer shuts down, the data in the memory is lost.

**Storage (SSD or Magnetic Disk)**  
Storage is used for keeping data over long periods of time. When a program runs, the CPU will direct the memory to temporarily load data from long-term storage.

**Network (LAN or the Internet)**  
Network is the gateway for anything that you need that isn't stored on your computer. The network could connect to other computers in the same room (a Local Area Network) or to a computer on the other side of the world, connected over the internet.

**Other Numbers to Know?**  
You may have noticed a few other numbers involving the L1 and L2 Cache, mutex locking, and branch mispredicts. While these concepts are important for a detailed understanding of what's going on inside your computer, you don't need to worry about them for this course. If you're curious to learn more, check out [Peter Norvig's original blog post](http://norvig.com/21-days.html) from a few years ago, and [an interactive version](http://people.eecs.berkeley.edu/~rcs/research/interactive_latency.html) for today's current hardware.

The CPU is the brains of a computer. The CPU has a few different functions including directing other components of a computer as well as running mathematical calculations. The CPU can also store small amounts of data inside itself in what are called **registers**. These registers hold data that the CPU is working with at the moment.

For example, say you write a program that reads in a 40 MB data file and then analyzes the file. When you execute the code, the instructions are loaded into the CPU. The CPU then instructs the computer to take the 40 MB from disk and store the data in memory (RAM). If you want to sum a column of data, then the CPU will essentially take two numbers at a time and sum them together. The accumulation of the sum needs to be stored somewhere while the CPU grabs the next number.

This cumulative sum will be stored in a register. The registers make computations more efficient: the registers avoid having to send data unnecessarily back and forth between memory (RAM) and the CPU.

Beyond the fact that memory is expensive and ephemeral, we'll learn that for most use cases in the industry, memory and CPU aren't the bottleneck. Instead the storage and network, which you'll learn about in the next videos, slow down many tasks you'll work on in the industry.

If a dataset is larger than the size of your RAM, you might still be able to analyze the data on a single computer. By default, the Python pandas library will read in an entire dataset from disk into memory. If the dataset is larger than your computer's memory, the program won't work.

However, the Python pandas library can read in a file in smaller chunks. Thus, if you were going to calculate summary statistics about the dataset such as a sum or count, you could read in a part of the dataset at a time and accumulate the sum or count.

[Here](http://pandas.pydata.org/pandas-docs/stable/user_guide/io.html#io-chunking) is an example of how this works.

**Hadoop Vocabulary**

Here is a list of some terms associated with Hadoop. You'll learn more about these terms and how they relate to Spark in the rest of the lesson.

* [Hadoop](https://hadoop.apache.org/) - an ecosystem of tools for big data storage and data analysis. Hadoop is an older system than Spark but is still used by many companies. The major difference between Spark and Hadoop is how they use memory. Hadoop writes intermediate results to disk whereas Spark tries to keep data in memory whenever possible. This makes Spark faster for many use cases.
* **Hadoop MapReduce** - a system for processing and analyzing large data sets in parallel.
* **Hadoop YARN** - a resource manager that schedules jobs across a cluster. The manager keeps track of what computer resources are available and then assigns those resources to specific tasks.
* **Hadoop Distributed File System (HDFS)** - a big data storage system that splits data into chunks and stores the chunks across a cluster of computers.

As Hadoop matured, other tools were developed to make Hadoop easier to work with. These tools included:

* **Apache Pig** - a SQL-like language that runs on top of Hadoop MapReduce
* **Apache Hive** - another SQL-like interface that runs on top of Hadoop MapReduce

Oftentimes when someone is talking about Hadoop in general terms, they are actually talking about Hadoop MapReduce. However, Hadoop is more than just MapReduce. In the next part of the lesson, you'll learn more about how MapReduce works.

**How is Spark related to Hadoop?**

Spark, which is the main focus of this course, is another big data framework. Spark contains libraries for data analysis, machine learning, graph analysis, and streaming live data. Spark is generally faster than Hadoop. This is because Hadoop writes intermediate results to disk whereas Spark tries to keep intermediate results in memory whenever possible.

The Hadoop ecosystem includes a distributed file storage system called HDFS (Hadoop Distributed File System). Spark, on the other hand, does not include a file storage system. You can use Spark on top of HDFS but you do not have to. Spark can read in data from other sources as well such as [Amazon S3](https://aws.amazon.com/s3/).

**Streaming Data**

Data streaming is a specialized topic in big data. The use case is when you want to store and analyze data in real-time such as Facebook posts or Twitter tweets.

Spark has a streaming library called [Spark Streaming](https://spark.apache.org/docs/latest/streaming-programming-guide.html) although it is not as popular and fast as some other streaming libraries. Other popular streaming libraries include [Storm](http://storm.apache.org/) and [Flink](https://flink.apache.org/" \t "_blank). Streaming won't be covered in this course, but you can follow these links to learn more about these technologies.

# MapReduce

MapReduce is a programming technique for manipulating large data sets. "Hadoop MapReduce" is a specific implementation of this programming technique.

The technique works by first dividing up a large dataset and distributing the data across a cluster. In the map step, each data is analyzed and converted into a (key, value) pair. Then these key-value pairs are shuffled across the cluster so that all keys are on the same machine. In the reduce step, the values with the same keys are combined together.

While Spark doesn't implement MapReduce, you can write Spark programs that behave in a similar way to the map-reduce paradigm. In the next section, you will run through a code example.

**Spark Use Cases and Resources**

Here are a few resources about different Spark use cases:

* [**Data Analytics**](http://spark.apache.org/sql/)
* [**Machine Learning**](http://spark.apache.org/mllib/)
* [**Streaming**](http://spark.apache.org/streaming/)
* [**Graph Analytics**](http://spark.apache.org/graphx/)

**You Don't Always Need Spark**

Spark is meant for big data sets that cannot fit on one computer. But you don't need Spark if you are working on smaller data sets. In the cases of data sets that can fit on your local computer, there are many other options out there you can use to manipulate data such as:

* [**AWK**](https://en.wikipedia.org/wiki/AWK) - a command line tool for manipulating text files
* [**R**](https://www.r-project.org/) - a programming language and software environment for statistical computing
* [**Python PyData Stack**](https://pydata.org/downloads.html), which includes pandas, Matplotlib, NumPy, and scikit-learn among other libraries

Sometimes, you can still use pandas on a single, local machine even if your data set is only a little bit larger than memory. Pandas can read data in chunks. Depending on your use case, you can filter the data and write out the relevant parts to disk.

If the data is already stored in a relational database such as [**MySQL**](https://www.mysql.com/) or [**Postgres**](https://www.postgresql.org/), you can leverage SQL to extract, filter and aggregate the data. If you would like to leverage pandas and SQL simultaneously, you can use libraries such as **[SQLAlchemy](https://www.sqlalchemy.org/" \t "_blank)**, which provides an abstraction layer to manipulate SQL tables with generative Python expressions.

The most commonly used Python Machine Learning library is **[scikit-learn](http://scikit-learn.org/stable/" \t "_blank)**. It has a wide range of algorithms for classification, regression, and clustering, as well as utilities for preprocessing data, fine tuning model parameters and testing their results. However, if you want to use more complex algorithms - like deep learning - you'll need to look further. [**TensorFlow**](https://www.tensorflow.org/) and **[PyTorch](https://pytorch.org/" \t "_blank)** are currently popular packages.

**Spark's Limitations**

Spark has some limitation.

Spark Streaming’s latency is at least 500 milliseconds since it operates on micro-batches of records, instead of processing one record at a time. Native streaming tools such as [**Storm**](http://storm.apache.org/), [**Apex**](https://apex.apache.org/), or **[Flink](https://flink.apache.org/" \t "_blank)** can push down this latency value and might be more suitable for low-latency applications. Flink and Apex can be used for batch computation as well, so if you're already using them for stream processing, there's no need to add Spark to your stack of technologies.

Another limitation of Spark is its selection of machine learning algorithms. Currently, Spark only supports algorithms that scale linearly with the input data size. In general, deep learning is not available either, though there are many projects integrate Spark with Tensorflow and other deep learning tools.

**Hadoop versus Spark**

The Hadoop ecosystem is a slightly older technology than the Spark ecosystem. In general, Hadoop MapReduce is slower than Spark because Hadoop writes data out to disk during intermediate steps. However, many big companies, such as Facebook and LinkedIn, started using Big Data early and built their infrastructure around the Hadoop ecosystem.

While Spark is great for iterative algorithms, there is not much of a performance boost over Hadoop MapReduce when doing simple counting. Migrating legacy code to Spark, especially on hundreds of nodes that are already in production, might not be worth the cost for the small performance boost.

**Beyond Spark for Storing and Processing Big Data**

Keep in mind that Spark is not a data storage system, and there are a number of tools besides Spark that can be used to process and analyze large datasets.

Sometimes it makes sense to use the power and simplicity of SQL on big data. For these cases, a new class of databases, know as NoSQL and NewSQL, have been developed.

For example, you might hear about newer database storage systems like [**HBase**](https://hbase.apache.org/) or [**Cassandra**](http://cassandra.apache.org/). There are also distributed SQL engines like [**Impala**](https://impala.apache.org/) and [**Presto**](https://prestodb.io/). Many of these technologies use query syntax that you are likely already familiar with based on your experiences with Python and SQL.

In the lessons ahead, you will learn about Spark specifically, but know that many of the skills you already have with SQL, Python, and soon enough, Spark, will also be useful if you end up needing to learn any of these additional Big Data tools.

**Quizzes:**



















