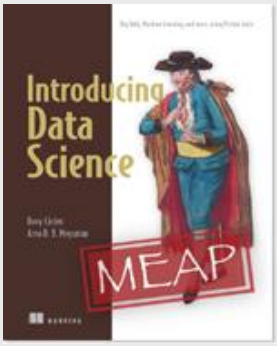

The big data ecosystem and data science

By Davy Cielen

The big data ecosystem can be grouped into technologies that have similar goals and functionalities. In this article, we'll explore those technologies.

This article is excerpted from Introducing Data Science. Save 39% on Introducing Data Science *with code 15dzamia at manning.com.*

Currently there are many big data tools and frameworks, and it is easy to get lost because new technologies appear rapidly. It becomes much easier once you realize that the big data ecosystem can be grouped into technologies that have similar goals and functionalities. Data scientists use many different technologies but not all of them. The mind map in figure 1 shows the components of the big data ecosystem and where the different technologies belong.

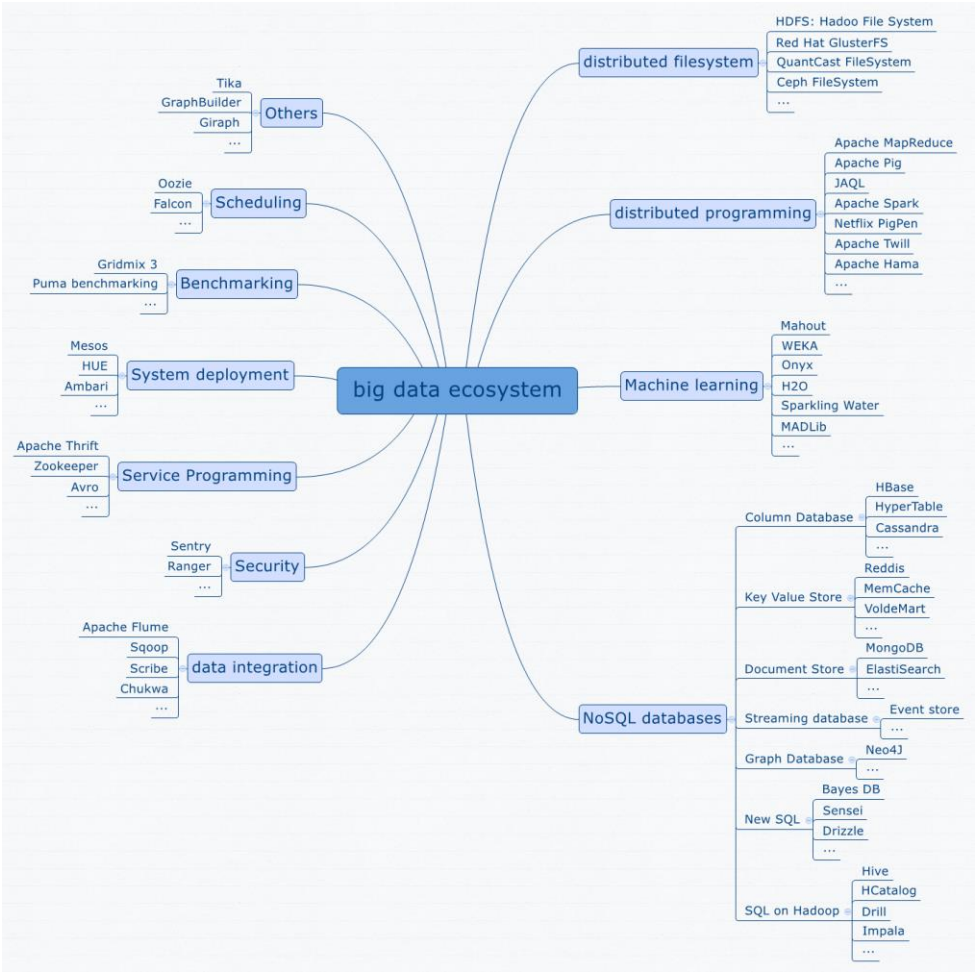

Figure 1 Big data technologies can be classified into a few main components.

Let's take a look at the different groups of tools in this diagram and see what each does. We'll start with distributed file systems, upon which everything else is built.

Distributed file systems

A *distributed file system* is similar to a normal file system except that it runs on multiple servers at once. Because it is a file system, you can do almost all the same things you would do on a normal file system. Actions such as storing, reading, and deleting files and adding

security to files are at the core of every file system, also the distributed one. Distributed file systems have some significant advantages:

- They can contain files larger than any one computer disk.
- Files get automatically replicated across multiple servers for redundancy or parallel operations while hiding the complexity of doing so from the user.
- The system scales easily, you are no longer bound by the memory or storage restrictions of a single server.

In the past, scale was increased by moving everything to a server with more memory, storage and a better CPU. Nowadays you simply add another small server, this principle makes the scaling potential virtually unlimited.

The best known distributed file system at this moment is the *Hadoop File System (HDFS)*. It is an open-source implementation of the Google File System. However, there are many other distributed file systems out there: *Red Hat Cluster FS*, *Ceph File System*, and *Tachyon File System*, to name but three.

Distributed programming framework

Once you have the data stored on the distributed file system, you want to exploit it. An important aspect of working on a distributed hard disk is that you will not move your data to your program, but rather you will move your program to the data. When you start from scratch with a normal general-purpose programming language such as C, Python, or Java, you need to deal with the complexities that come with distributed programming such as restarting jobs that have failed, tracking the results from the different subprocesses, and so on. Luckily, the open-source community has developed many frameworks to handle this for you and give you a much better experience working with distributed data and dealing with many of the challenges it carries.

Data integration framework

Once you have a distributed file system in place, you need to add some data. This means that you need to move data from one source to another, and this is where the data integration frameworks such as Apache Sqoop and Apache Flume excel. The process is similar to an extract, transform, and load process in a traditional data warehouse.

Machine learning frameworks

When you have the data in place, it's time to extract the coveted insights. This is where you rely on the fields of machine learning, statistics, and applied mathematics. Before World War II, everything needed to be calculated by hand, which severely limited the possibilities of data analysis. After World War II computers and scientific computing were developed. A single computer could do all the counting and calculations and a world of opportunities opened. Ever

since this breakthrough, people just need to derive the mathematical formulas, write them in an algorithm, and load their data.

With the enormous amount of data available nowadays, one computer can no longer handle the workload by itself. In fact, some of the algorithms developed in the previous millennium would never terminate before the end of the universe, even if you could use every computer available on earth.

One of the biggest issues with the old algorithms is that they do not scale well. With the amount of data we need to analyze today, this becomes problematic, and specialized frameworks and libraries are required in order to deal with this amount of data. The most popular machine learning library for python is **Scikit-learn**, it's a great machine learning toolbox and we will be using it later in the book. There are off course other python libraries:

PyBrain for neural networks. Neural networks are learning algorithms that essentially mimic the human brain in learning mechanic and complexity. Neural networks are often regarded advanced and black box.

Nltk or Natural Language Toolkit. As the name suggest its focus is working with natural language. It's an extensive library that comes bundled with a number of text corpuses to help you model your own data.

Pylearn2. Another machine learning toolbox but a bit less mature than Scikit-learn.

The landscape doesn't end with python libraries of course. **Spark** is a new Apache-licensed machine learning engine, specialized at real learn-time machine learning. It's worth taking a look at and you can read more about it here: http://spark.apache.org/

NoSQL databases

If you need to store huge amounts of data, you require software that is specialized in managing and querying this data. Traditionally this has been the playing field of relational databases such as Oracle SQL, MySQL, Sybase IQ, and others. While they still are the go-to technology for many use cases, new types of databases have emerged under the grouping of NoSQL databases.

The name of this group can be misleading as "No" in this context stands for "Not Only." A lack of functionality in SQL is not the biggest reason for the paradigm shift, and many of the NoSQL databases have implemented a version of SQL themselves. But traditional databases had some shortcomings that did not allow them to scale well. By solving some of the problems of traditional databases, NoSQL databases allow for a virtually endless growth of data.

Many different types of databases have arisen, but they can be categorized into the following types:

- *Column databases*—Data is stored in columns, which allows some algorithms to perform much faster queries. Newer technologies use cell-wise storage. Table-like structures are

still very important.

- *Document stores*—Document stores no longer use tables but store every observation in a document. This allows for a much more flexible data scheme.
- *Streaming data*—Data is collected, transformed, and aggregated not in batches but in real time. Although we have categorized it here as a database to help you in tool selection, it is more a particular type of problem that drove creation of technologies like Storm.
- *Key-value stores*—Data is not stored in a table; rather you assign a key for every value such as org.marketing.sales.2015: 20000. This scales very well but places almost all the implementation on the developer.
- *SQL on Hadoop*—Batch queries on Hadoop are in a SQL-like language that uses the map-reduce framework in the background.
- *New SQL*—This class combines the scalability of NoSQL databases with the advantages of a relational database. They all have a SQL interface and a relational data model.
- *Graph databases*—Not every problem is best stored in a table. Some problems are more naturally translated into graph theory and stored in graph databases. A classic example of this is a social network.

Scheduling tools

Scheduling tools help you to automate repetitive tasks and trigger jobs based on events such as adding a new file to a folder. These are similar to tools like CRON on Linux but specifically developed for big data. You can use them, for instance, to start a map-reduce task whenever a new dataset is available in a directory.

Benchmarking tools

This class of tools was developed to optimize your big data installation by providing standardized profiling suites. A profiling suite is taken from a representative set of big data jobs. Benchmarking and optimizing the big data infrastructure and configuration are not often jobs for data scientists themselves but for a professional specialized in setting up IT infrastructure. Using an optimized infrastructure can make a big cost difference. For example, if you can gain 10% on a cluster of 100 servers, you save the cost of 10 servers.

System deployment

Setting up a big data infrastructure is not an easy task and assisting engineers in deploying new applications into the big data cluster is where system deployment tools shine. They largely automate the installation and configuration of big data components. This is not a core task of a data scientist.

Service programming

Suppose that you have made a world-class soccer prediction application on Hadoop, and you want to allow others to use the predictions made by your application. However, you have no idea of the architecture or technology of everyone keen on using your predictions. Service tools excel here by exposing big data applications to other applications as a service. Data scientists sometimes need to expose their models through services. The best known example is the REST service were **REST** stands for representational state transfer. It is often used to feed websites with data.

Security

Do you want everybody to have access to all of your data? If so, you probably need to have fine-grained control over the access to data but don't want to manage this on an application-by-application basis. Big data security tools allow you to have central and fine-grained control over access to the data. Big data security has become a topic in its own right, and data scientists will usually only be confronted with it as a data consumer, seldom will they implement the security themselves.