CHAPTER 1

Big Data, Big Challenges

In this chapter, we expose the modern architecture challenges facing the SMACK stack (Apache Spark, Mesos, Akka, Cassandra, and Kafka). Also, we present dynamic processing environment problems to see which conditions are suitable and which are not.

This chapter covers the following:

- Why we need a pipeline architecture for big data
- The Lambda Architecture concept
- ETL and its dark side

Big Data Problems

We live in the information era, where almost everything is data. In modern organizations, there is a suitable difference between data engineers and data architects. Data engineers are experts who perfectly know the inner workings and handling of the data engines. The data architect well understands all the data sources—internal and external. Internal sources are usually owned systems. External sources are systems outside the organization. The first big data problem is that the number of data sources increases with time.

A few years ago, a big company's IT department could survive without data architects or data engineers. Today's challenge is to find good architects. The main purpose of architecture is always resilience. If the data architect doesn't have a data plan, then the data sources and the data size will become unmanageable.

The second problem is obtaining a data sample. When you are a data analyst (the person charged with the compilation and analysis of numerical information), you need data samples—that is, data from production environments. If the size of the data and/or the number of data sources increases, then obtaining data samples becomes a herculean task.

The third big data problem is that the validity of an analysis becomes obsolete as time progresses. Today, we have all the information. The true value of data is related to time. Within minutes, a recommendation, an analysis, or a prediction can become useless.

The fourth problem is related to the return on investment of an analysis. The analysis velocity is directly proportional to the return on investment. If the data analyst can't get data in a timely way, then analysis costs increase and the earnings decrease.

Electronic supplementary material The online version of this chapter (doi:10.1007/978-1-4842-2175-4_1) contains supplementary material, which is available to authorized users.

R. Estrada and I. Ruiz, *Big Data SMACK*, DOI 10.1007/978-1-4842-2175-4_1

Infrastructure Needs

Modern companies require a scalable infrastructure. The costs of your data center are always in accordance with your business size. There is expensive hardware and costly software. And nowadays, when it comes to open source software, people's first thoughts are the high costs of consulting or the developer's price tag. But there is good news: today, big data solutions are not exclusive to large budgets.

Technologies must be distributed. Nowadays, when we talk about *distributed software*, we are no longer talking about multiple processors; instead, we are talking about multiple data centers. This is the same system, geographically dispersed.

If your business grows, your data should fit those needs. This is scalability. Most people are afraid of the term *big data*, and spend valuable economic resources to tackle a problem that they don't have. In a traditional way, your business growth implies your data volumes' growth. Here, the good news is scale linearly with cheap hardware and inexpensive software.

Faster processing speed is not related to processor cycles per second, but the speed of all your enterprise process. The now is everything, opportunities are unique, and few situations are repeatable.

When we talk about complex processing, we are not talking about the "Big O" of an algorithm. This is related to the number of actors involved in one process.

The data flow is constant. The days when businesses could store everything in warehouses are gone. The businesses that deliver responses the next day are dying. The *now* is everything. Data warehouses are dying because stored data becomes rotten, and data caducity is shorter every day. The costs associated with a warehouse are not affordable today.

And finally, there is visible and reproducible analysis. As we have mentioned, data analysts need fresh and live data to satisfy their needs. If data becomes opaque, the business experiences a lack of management.

ETL

ETL stands for *extract, transform, load*. And it is, even today, a very painful process. The design and maintenance of an ETL process is risky and difficult. Contrary to what many enterprises believe, they serve the ETL and the ETL doesn't serve anyone. It is not a requirement; it is a set of unnecessary steps.

Each step in ETL has its own risk and introduces errors. Sometimes, the time spent debugging the ETL result is longer than the ETL process itself. ETL always introduces errors. Everyone dedicated to ETL knows that having no errors is an error. In addition, everyone dedicated to ETL knows that applying ETL onto sensitive data is playing with the company's stability.

Everybody knows that when there is a failure in an ETL process, data duplication odds are high. Expensive debugging processes (human and technological) should be applied after an ETL failure. This means looking for duplicates and restoring information.

The tools usually cost millions of dollars. Big companies know that ETL is good business for them, but not for the client. The human race has invested a lot of resources (temporal and economic) in making ETL tools.

The ETL decreases throughput. The performance of the entire company decreases when the ETL process is running, because the ETL process demands resources: network, database, disk space, processors, humans, and so forth.

The ETL increases complexity. Few computational processes are as common and as complicated. When a process requires ETL, the consultants know that the process will be complex, because ETL rarely adds value to a business's "line of sight" and requires multiple actors, steps, and conditions.

ETL requires intermediary files writing. Yes, as if computational resources were infinite, costless, and easily replaceable. In today's economy, the concept of big intermediary files is an aberration that should be removed.

The ETL involves parsing and reparsing text files. Yes, the lack of appropriate data structures leads to unnecessary parsing processes. And when they finish, the result must be reparsed to ensure the consistency and integrity of the generated files.

Finally, the ETL pattern should be duplicated over all our data centers. The number doesn't matter; the ETL should be replicated in every data center.

The good news is that no ETL pipelines are typically built on the SMACK stack. ETL is the opposite of high availability, resiliency, and distribution. As rule of thumb, if you write a lot of intermediary files, you suffer ETL; as if your resources—computational and economic—were infinite.

The first step is to remove the extract phase. Today we have very powerful tools (for example, Scala) that can work with binary data preserved under strongly typed schemas (instead of using big text dumps parsed among several heterogeneous systems). Thus, it is an elegant weapon for a more civilized big data age.

The second step is to remove the load phase. Today, your data collection can be done with a modern distributed messaging system (for example, Kafka) and you can make the distribution to all your clients in real time. There is no need to batch "load."

Lambda Architecture

Lambda Architecture is a data processing architecture designed to handle massive quantities of data by taking advantage of both batch and stream processing methods. As you saw in previous sections, today's challenge is to have the batch and streaming at the same time.

One of the best options is Spark. This wonderful framework allows batch and stream data processing in the same application at the same time. Unlike many Lambda solutions, SMACK satisfies these two requirements: it can handle a data stream in real time and handle despair data models from multiple data sources.

In SMACK, we persist in Cassandra, the analytics data produced by Spark, so we guarantee the access to historical data as requested. In case of failure, Cassandra has the resiliency to replay our data before the error. Spark is not the only tool that allows both behaviors at the same time, but we believe that Apache Spark is the best.

Hadoop

Apache Hadoop is an open-source software framework written in Java for distributed storage and the distributed processing of very large data sets on computer clusters built from commodity hardware.

There are two main components associated with Hadoop: Hadoop MapReduce and Hadoop Distributed File System (HDFS). These components were inspired by the Google file system.

We could talk more about Hadoop, but there are lots of books specifically written on this topic. Hadoop was designed in a context where size, scope, and data completeness are more important than speed of response.

And here you face with a crucial decision: if the issue that you need to solve is more like data warehousing and batch processing, Apache Hadoop could be your solution. On the other hand, if the issue is the speed of response and the amount of information is measured in speed units instead of data size units, Apache Spark is your solution.

Data Center Operation

And we take this space to briefly reflect on how the data center operation has changed.

Yesterday, everything scaled up; today, everything scales out. A few years ago, the term *data center* meant proprietary use of specialized and expensive supercomputers. Today's challenge is to be competitive using commodity computers connected with a non-expensive network.

The total cost of ownership determines all. Business determines the cost and size of the data center. Modern startups always rise from a small data center. Buying or renting an expensive data center just to see if your startup is a good idea has no meaning in the modern economy.

The M in SMACK is a good solution to all your data center needs. With Apache Mesos, you can "abstract" all the resources from all the interconnected small computers to build a supercomputer with the linear sum of each machine's resources: CPU cores, memory, disk, and network.

The Open Source Reign

A few years ago, dependency on a vendor was a double-edged sword. On one hand, large companies hired proprietary software firms to later blame the manufacturer for any failure in their systems. But, on the other hand, this dependence—all the processes, development, and maintenance—became slow and all the issues were discussed with a contract in hand.

Many large companies don't implement open source solutions for fear that no one else can provide the same support as large manufacturers. But weighing both proposals, the vendor lock-in and the external bug fixing is typically more expensive than open source solutions.

In the past, the big three-letter monopolies dictated the game rules. Today, the rules are made "by and for" the developers, the transparency is guaranteed by APIs defined by the same community. Some groups—like the Apache Software Foundation and the Eclipse Foundation—provide guides, infrastructure, and tools for sustainable and fair development of these technologies.

Obviously, nothing is free in this life; companies must invest in training their staff on open source technologies.

The Data Store Diversification

Few people see this, but this is the beginning of the decline of the relational databases era. Since 2010, and the emergence of NoSQL and NoETL, there has been tough criticism of traditional systems, which is redefining the leader board.

Due to modern business needs, having everything stored in a relational database will go from being the standard way to the old-fashioned and obsolete way. Simple daily problems like recording the data, multiple store synchronization, and expensive store size are promoting NoSQL and NoETL solutions.

When moving data, gravity and location matter. Data gravity is related to the costs associated with moving a huge amount of data from one point to another. Sometimes, the simple everyday task of restoring a backup can be a daunting task in terms of time and money.

Data allocation is a modern concept related to moving the computation resources where the data is located, rather than moving the data to where the computation is. It sounds simple, but due to the hardware (re)evolution, the ability to perform complex calculations on new and powerful client machines doesn't impact customer perception on the performance of the entire system.

DevOps (development operations) is a term coined by Andrew Clay Shafer and Patrick Debois at the Agile Conference in 2008.[1] Since then, DevOps has become a movement, a culture, and a lifestyle where software developers and information technology professionals charged with data center operation can live and work in harmony. How is this achieved? Easy: by dissolving the differences between them.

Today DevOps is one of the most profitable IT specializations. Modern tools like Docker and Spark simplify the movement between testing and production environments. The developers can have production data easily and the testing environments are almost mirrored with production environments.

As you will see in Chapter 7, today's tendency is containerize the development pipeline from development to production.

[1] http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=4599439

Is SMACK the Solution?

Even today, there are very few companies fully using SMACK. That is, many major companies use a flavor of SMACK—just use one, two, or three letters of the SMACK stack. As previously mentioned, Spark has many advantages over Hadoop. Spark also solves problems that Hadoop cannot. However, there are some environments where Hadoop has deep roots and where workflow is completely batch based. In these instances, Hadoop is usually a better choice.

Several SMACK letters have become a requirement for some companies that are in pilot stages and aim to capitalize all the investment in big data tools and training. The purpose of this book is to give you options. The goal is not to make a full pipeline architecture installation of all the five technologies.

However, there are many alternatives to the SMACK stack technologies. For example, Yarn may be an alternative to Mesos. For batch processing, Apache Flink can be an alternative to Spark. The SMACK stack axiom is to build an end-to-end pipeline and have the right component in the correct position, so that integration can be done quickly and naturally, instead of having expensive tools that require a lot of effort to cohabit among them.