Introduction to Big Data

Isaac Triguero*

February 3, 2022

1 What is Big Data?

'Everybody' is talking about Big Data. Many different fields such as finances, physics, biology, medicine and many more are gathering loads of information that they hope to utilise in some way to extract knowledge.

But when it comes to answering the question of what is Big Data, people start talking about gigabytes of data, or terabytes of data, focusing on the volume, but the thing is that there is not a standard definition of what Big Data actually is. For that reason, some people tend to call Big Data to anything that could be somehow no more than standard machine learning and data mining problems.

It is actually impossible to define Big Data only based on a specific amount of data. Although there isn't a standard definition I very much like this one:

"Big Data involves data whose volume, diversity and complexity requires new techniques, algorithms and analyses to extract valuable knowledge (hidden)"

So, if you ask me, are we talking about 20 megabytes, 8 gigabytes, 100 terabytes? My answer will be: 'It depends'.

It does depend on what you want to do with that data, the kind of analysis you need to perform on the data to obtain that knowledge you are seeking. In layman terms, if you can do that on your laptop, that's not big data! You will have to use distributed solutions to handle that data. I also like to refer to Big Data as *data intensive* applications.

It is also important to remember that what we call Big Data today, might be almost nothing tomorrow, as the computer power available to us is growing every day as well.

2 Success (and failure) of Big Data

I want to start off with a good example of success (and also failure) of 'real' Big Data.

A while ago, there was a fear of a pandemic due to a new strand of the Swine Flu (H1N1). At the time, Statistical Centres took a bit of time to process medical data about this disease (1-2 weeks delay). Google soon realised of that and came up with a very innovate solution.

They used query data, that is, searches made on Google, and they found a strong correlation between the real historical data about location of the disease and the terms that people were using in their searches. Thus, they developed a solution Google Flue Trends (GFTs) that allowed to monitor the spread of the Swine flu in real time (they make up for these 1-2 weeks of delay). More

*In collaboration with Dr Mikel Galar (UPNA)

info in the paper published in Nature in 2009: Detecting influenza epidemics using search engine query data.

But, why is this Big Data?

They certainly needed to parallelise their data analytics tools to handle more than 3,000 million searches every day, comparing most common 50 million search terms against a database of flu propagation from 2003 to 2008. But again, it is not about the volume, but the need to parallise their analyses. In particular, they used a solution based on Hadoop MapReduce.

You might be thinking, wow, so Google made a real-time solution for pandemics... why didn't it work with COVID though? They actually stopped GFTs in 2014 after they overestimated the flu levels in 2013. Various reasons for this to fail, including how people changed their behaviour around their searches, and the influence of the media when talking about this. Machine learning and data mining models need to be updated with new data and constantly adapt the models; this is usually called data stream mining.

In you want to know more about this, please read the paper: The parable of Google Flu: Traps in Big Data Analysis

I just wanted to show you this example, as a representative case of success in Big Data, but at the same time, exemplify the difficulties of doing Big Data analytics, a relatively young field in Computer Science that changes extremely quickly.

3 Big Data Evolution and Definition

Our world is revolving around data. Science, business, medicine, industry, energy, anything you can imagine, they all generate and store data. At the beginning of 2020, the digital universe was estimated to consist of 44 zettabytes of data (According to `seedscientific.com`). I don't really know how accurate that figure is, but certainly, we do generate way too much data, and going from analog to digital storage has immensely increased our ability to store it.

Somehow we are under the impression that data is the new gold. The advent of the Internet of Things has landed to stay here with us, and we must take advantage of it to turn it into profitable products. But, is it all about volume? Shall we just collect and store data for the sake of it? I will get back to this later.

Although I told you that there isn't a formal/standard definition of what Big Data is, already in 2001, Gartner defined/characterised Big Data using the three different V's: Volume, Velocity and Variety. Have a look at: D. Laney, 3D Data Management: Controlling Data Volume, Velocity, and Variety. Gartner file.

- The **Volume**, what they call *data at rest*, is the obvious one. We have seen an increase in the amount of data that is being generated every second. In data mining, datasets have seen an enormous increase in the number of examples and features. We are expanding from gigabytes to petabytes of information, and unfortunately it's not like you could currently have a 1 petabyte drive on your laptop/desktop computer to process it.

- The **Velocity**, known as *data in motion*, which means the speed at which the data comes to us, and also the speed at which we have to process the data if we want to take advantage of it! We are moving from traditional batch applications to real time. Late decisions may imply missing opportunities.

- The **Variety**, *data in many forms*, meaning that we might generate/use many kinds of data with different formats and structure: (1) traditional structured data/tables such as tables

and relational databases for which we know the schema, (2) semi-structure data such as XML files for which we still can infer the schema, or (3) completely unstructured data such as text, images, audio, video for which we don't know the schema and we might want to impose some structure prior to doing any analyses.

Big Data is expanding on these three fronts at an increasing rate.
Others have expanded this 3 V's definition to 5 V's, including: **Veracity** and **Value**.

- The **Veracity**, *data in doubt*, refers to the problem of trusting your data. We have so much data coming from different sources that there is uncertainty about the quality of that data. Data that may be missing, be ambiguous, or simply put, be wrong.
- Anyway, all of these V's are useless without the central V, the **Value** of it, *data in use*. With bigger datasets we (supposedly) have an opportunity to find better patterns and better insights about the problem. Although, you will read from me later saying that more data doesn't necessarily imply better results.

Many researchers have become very creative, and you will find on-line additional V's (I lost count at 10). But I guess these 5 really give you a flavour of what we are talking about here.

The world of Big Data has many different faces such as databases, security and privacy, visualisation, computational infrastructure or data analytics/mining. In this module we are interested in the analytics/mining part, and how to extract knowledge from big data sets.

4 How to deal with Big Data?

So, let's assume we embark ourselves into exploring 100 terabytes of data. A single machine wouldn't be able to store that data to begin with, and of course it would lack the processing capabilities to process so much data in a reasonable amount time.

What do we do? What is the solution? Well, no surprise here: *divide-and-conquer*.

Rather than trying to make a single machine more powerful by adding more RAM memory, getting a bigger hard drive, etc, which by the way could be actually very expensive and the improvements could not (and will not) be sufficient, if we were to have more than one computer, why don't we try to spread the processing across them?

Adding more resources to a single computer is what we call *vertical scalability* or *scale-up*, as opposed to *horizontal scalability* or *scale-out*, which involves using more than one computer.

The good thing about using multiple machines to address the problem is that they may not need specific hardware (i.e. high-end specs) to tackle Big Data, so they can be cheaper, and if we need more computing power we just simply add more computers.

However, this horizontal scalability is not free of issues. For starters, the information goes through a network which may become a bottleneck, and the design of the software will become more complex because there is a need to coordinate actions across computing nodes. But there are many other issues we should bear in mind: we will need more space, energy costs (electricity and cooling systems), cost of the network equipment, software licenses, etc.

One more important factor that makes horizontal scalability very appealing is the possibility of making our applications *fault-tolerant*. If you have a single very powerful machine that for whatever reason crashes, this will be a problem in processing your Big Data, especially if this crashes after several days! However, having multiple machines may allow us to provide some

fault-tolerance, and some other machines may pick up on the work that a machine that breaks was doing.

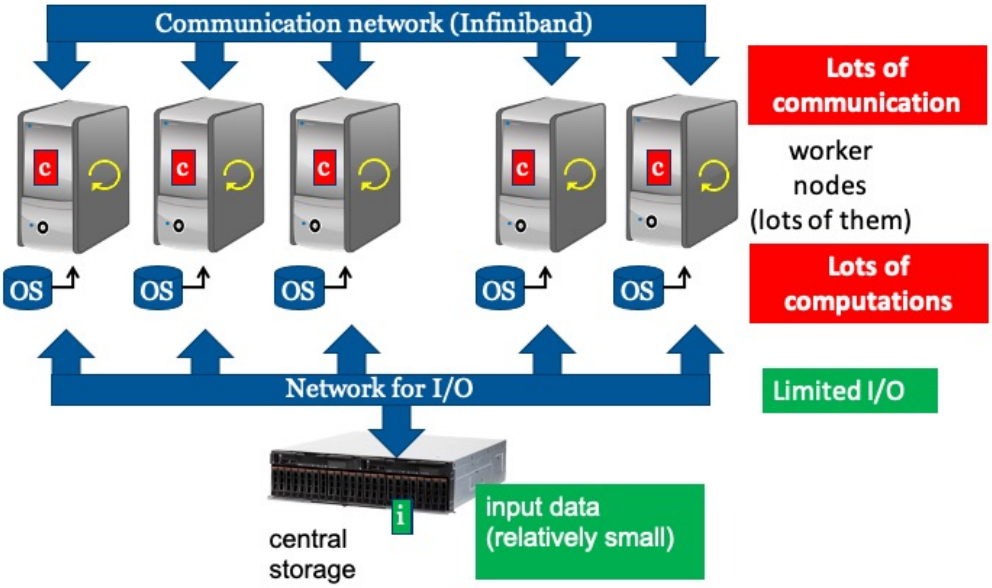
Hope this helps you understand why Big data technologies are based on *horizontal scalability* to be able to handle the different V's that characterise Big Data.

4.1 High Performance computing vs Big Data Computing

If you know what High Performance Computing (HPC) is, you're probably now thinking, ok yes, if I am going to deal with Big Data, I will have to use that kind of facility and not just my laptop.

If you don't know what an HPC is, this is simply a cluster of independent computers (nodes) that are connected together, using a communication network (e.g. Infiniband if you're lucky) and they usually have access to a central storage. Each of the nodes has one or more multi-core CPUs, their own RAM memory and a hard disk. In an HPC setting, these nodes typically run a Linux operation system. It is important to note that software that runs on a node can access the RAM of that node, but not the RAM of another node. This is called a *distributed-memory* architecture.

Traditionally, HPCs have been used to tackle problems that require lots of computation but relatively small data, so all nodes can have access to the data and the distributed solutions spread that computation across the nodes. Usually, the different subtasks tackled by each node are not independent, so there is a need for communication of intermediate results (i.e. calculations). However, those 'messages' are usually not 'very big'. As a remark, the different nodes usually have their own local storage, but not intensively used during job execution.



Source: Jan Fostier. Introduction to MapReduce and its Application to Post-Sequencing Analysis

So, we have a lot of computers connected through a network, how do we program a distributed program?

We need to take into account that network communication. We usually use a driver node that is going to coordinate everything and a set of worker nodes that perform the actual calculations. One of the classical ways of dealing with distributed computing is with MPI (Message

Passing Interface). Without going much into detail, this is based on two main functions: scatter and gather. The former sends a message to all computing nodes that perform an operation and return a response. The latter receives/collects compute from the worker nodes.

One of the things I don't like much about MPI, is that you need to explicitly program that communication, and determine what each node is going to do. The main advantage of the technologies we are going be looking at in this course is that they will abstract the details of that network communication. They are then transparent to the programmer, but we still need to know what's going on to do this efficiently!

Another big disadvantage of this programming style is that they are not fault-tolerant, hence, not very appropriate for Big Data.

But, these are not the main reasons for me to distinguish between a standard HPC and a Big Data Cluster. Why is an HPC still not appropriate for Big Data?

In Big Data, application will devote most of their time processing and manipulating the data, which results in lots of time spent in Inputs/Outputs, which is moving data across the network, which is a very slow process.

Applying the HPC techniques to big data problems doesn't scale, simply because the central storage and/or I/O network cannot deliver the data to the worker nodes at a sufficient rate to keep the worker nodes busy.

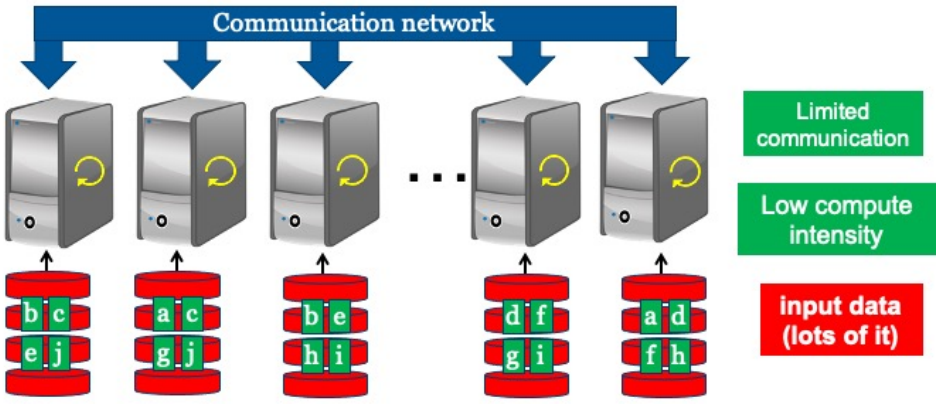
In summary, Big Data is not HPC, they are different computation models and we need a different hardware architecture to run on.

What do we have to do differently tackle data intensive jobs?

The limiting factor in a HPC is on the communication network, so, it is crucially important to minimise data movement across the network. Therefore, storing data in a central place and moving all data to the compute nodes when executing a job yields only poor performance. Cluster architectures for Big Data solutions should ensure that data and compute power are co-located.

In other words, the data should reside very close to where to actual computations will take place. As mentioned before, in HPC local drives are not really being used, how about using them to keep a fraction of the data in every single node?

This is referred to as the principle of **data locality**. Data locality is what distinguishes Big Data clusters from HPC clusters. To provide fault tolerance, there is a need for some data replication across computing nodes, so that, if one node fails, there still another node capable of handling that part of the data.



Solution: store data on local disks of the nodes that perform computations on that data ("**data locality**")

The following table summarises the key differences between Big Data and HPC:

Big Data	HPC
Focus on data-intensive jobs	Focus on computation-intensive jobs
H/W failure common	Surprised by H/W failure
Code: graphs, data mining	Code: simulation, optimisation
Usually mix CPU and data	Mix CPU/GPU
Job is moved to where the data is located	Data is moved to where it will be processed.
Single Instruction Multiple Data model (SIMD): Data parallelism	SIMD/MIMD model (more general parallelism)
Ok with Commodity hardware	Need specialised hardware

4.2 Distributed Systems for data-intensive jobs

I hope that is clear by now that the main objective in Big Data is to apply an operation/task to all data and we assume that one machine cannot process or store all of it.

What do want to do? we would like to use multiple computers and the data will be distributed across them. You don't really want to care much about which specific machine is executing the operation on a fraction of the data as soon as you get the result. Likewise, if there is any failure in one particular machine (or even worse, straggler nodes), you still want the original task to be completed, even if that means that part of the processing needs to be repeated in a different node.

If you allow me to put one more thing on the wish list to develop distributed systems, I wouldn't like to know much detail about which node is doing what. I look for an abstraction of the complexity behind distributed systems.

As said before, the principal of data locality must always be taken into consideration, avoiding data transfers between machines as much as possible.

To do so, we are going to introduce a new Programming paradigm called MapReduce which is based on the idea of:

"Moving computation is cheaper than moving computation and data at the same time"

In summary, we will distribute the data among nodes using a distributed file system. Functions/operations to process data are distributed to all the computing nodes, and each computing node works with the data stored in it locally! Only the necessary data should be moved across the network.

In the next chapter we will introduce this new programming paradigm.