UNIVERSITY OF TRIESTE

Department of Mathematics and Geosciences

Master Degree in Data Science and Scientific Computing

Your main title

Smaller subtitle that expands the main title

Thesis Coordinator:

...

Student:

Nicolas Solomita

Thesis Advisor:

...

Academic Year 2020/2021

Abstract

Keywords – Template, NHH, master thesis, LaTeX

Contents

List of Figures

List of Tables

Introduction

dfgjhkl

1 Streaming data

1.1 The world of big data

Although the concept of big data itself is relatively new, the origins of large data sets go back to the '60s when the world of data was just getting started with the first data centers and the development of relational databases.

Around 2005, people began to realize how much data were generated by the users through social networks and other online services. Hadoop, that is an open-source framework created specifically to store and analyze big data sets, was developed in 2006 and NoSQL technologies also began to gain popularity during that time. In the years since then, the volume of data has skyrocketed thanks to all sort of devices and sensors that started to generate tons of signals and information.

However, while big data has come far, its usefulness is only at the beginning. For instance, cloud computing has expanded big data possibilities even further thanks to the truly elastic scalability it can provide in a cheaper and fast way. [1]

Now the question is: **what exactly is big data?** There is not a unified and unique definition for this concept. In a general setting, big data refers to massive complex structured and unstructured data sets that are rapidly generated and transmitted from a wide variety of sources. This is also known as the 3Vs concept [2], even though over the years people started to add "Vs" in order to better categorize the notion.

1.1.1 The "Vs" of big data

Let's list some important concepts to better characterize the notion of big data (Fig. 1.2).

Volume Large volumes of data are generated in this very moment: it could be a few tens of terabytes for some organizations, or several hundred petabytes for others. The point is that sometimes the volume is so massive that cannot be stored on its entirety, but has to be compressed and transformed online. Even though traditional RDBMS could be used, it would be too expensive in terms of cost or time.

Velocity It is the fast rate at which data is received and maybe acted on. If the data is sent directly to memory, rather than written to disk, the speed will be higher and you will provide data in (near) real-time. For companies that rely upon fast-generated data, it is also important to analyze them as fast as possible, in order to take decisions on the fly.

Variety It refers to the many types of data that are available. Traditional data types were structured and fit neatly in a relational database. With the rise of big data, there are new unstructured and semistructured data types (as shown in Fig. 1.1) that require additional preprocessing in order to extrapolate meaning from them.

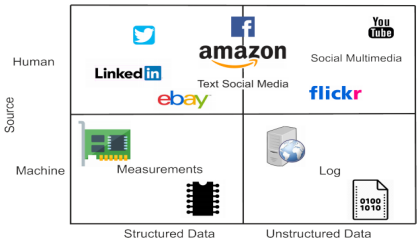

Fig. 1.1: Big data types

Veracity You can collect a lot of data from social media or websites, but how can you be sure everything is correct and truthful? Poor quality data not verified can cause problems, leading to inaccurate analysis and bad decisions (*garbage in = garbage out*). Therefore, you should always check your data in order to get valid and meaningful results.

Value Data has intrinsic value, but not all the collected data has real business value or benefits. As a result, organizations need to confirm that data relates to relevant business issues before using it in big analytics projects.

Variability When you have a lot of data, you can use it for multiple purposes and format it in different ways. Variability is the ability to use the very same data in multiple scenarios getting precise answers from it.[1][3][4]

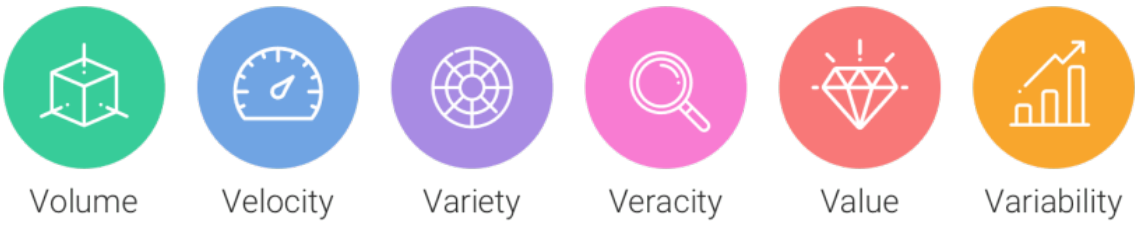

Fig. 1.2: 6Vs of big data

do we really need big data?????

1.2 Batch vs streaming

The distinction between batch processing and stream (or real-time) processing is one of the most fundamental principles within the big data world. As usual, even though there is no official definition of these two terms, a brief description is given in the following paragraphs.

Batch processing A batch is a collection of data points that have been grouped together within a specific time interval. As shown in Fig. 1.3, a good metaphor is a bucket of water: you use the water as soon as the bucket is full. This model is often used in situations where you do not need real-time analytics results or when it is more important to process large volumes of information in order to get a greater level of detail, even waiting a little bit longer. Since batch data needs to be loaded into some type of storage in order to be processed and analysed, this approach works well also in dealing with data sources from legacy systems, where it is not feasible to deliver data in streams.

Fig. 1.3: Bucket

Moreover, batch analytics remains as useful today as ever thanks to the changes brought by the rise of cloud technologies. These approaches have revolutionized the way all types of processing work by allowing data from many kinds of sources to be merged and integrated seamlessly and stored remotely. A simple example is the fundamental ETL process, that could be itself a batch job and it gained a lot from the advent of the cloud.

Stream processing Gartner Glossary states: *"Real-time analytics is the discipline that applies logic and mathematics to data to provide insights for making better decisions quickly"*.[5] More specifically, stream processing is the analysis of huge pools of current and in-motion data through the use of continuous queries, called event streams. These streams are triggered by a specific event that happens as a direct result of an action or set of actions. Under the streaming model, data is fed into analytics tools piece-by-piece as soon as it is generated. This time, as shown in Fig. 1.4, a good metaphor is a faucet from which the water comes out continuously. By using this approach,

Fig. 1.4: Faucet

organizations can extract business value from data in motion just like traditional analytics tools would allow them to do with data at rest.

Real-time analytics advantages could be:

- **Data visualization**: keeping an eye on the most important company information can help organizations manage their KPIs on a daily basis.
- **Business insights**: in case an out of the ordinary business event occurs, it will first show up in the relevant dashboard and this is fundamental where abnormal behavior should be flagged for investigation right away.
- **Increased competitiveness**: businesses looking to gain a competitive advantage can use streaming data to discern trends and set benchmarks faster in order to outpace their competitors.
- **Cutting preventable losses**: streaming analytics can prevent, or at least reduce, the damage of incidents like security breaches, manufacturing issues, customer churn, stock exchange meltdowns or social media crisis.

Both models are valuable and each of them can be used to address different use cases. Although streaming processing is best for use cases where time matters and batch processing works well when all the data has been collected, it is not a matter of which one is better than the other; it really depends on the business objective. In some cases, the same company may employ even both processes together in the same system.

Moreover, nowadays it is possible to use specific tools in order to turn batch data into streaming data. This because enterprises are shifting priorities towards real-time analytics and data streams to glean actionable information in real time. However, a good takeaway could be: *"just because you have a hammer, it does not mean that is the right tool for the job"*. Batch and streaming processing are two different models and it is not a matter of choosing one over the other, it is about being smart and determining which one is better for the specific use case. A brief recap of the main characteristics is present in Table 1.1. [6][7][8]

	Batch Processing	**Streaming Processing**
Analytics	Complex analytics on all or most of the data	Simple response functions on rolling window or most recent data
Size	Large batches	Micro batches
Performance	High latency / High delay	Low latency / Low delay

Table 1.1: Batch vs Stream processing

1.3 Streaming analytics challenges

Building robust stream processing systems is hard, because there are a number of problems that rise when we talk about data in motion. In this section, some of the problems, that were taken into account during the entire project, are addressed. Everything is focused on the general solutions, rather than explaining how the specific technologies handle them, this being the central point of the next chapters.

1.3.1 Data and operations complexity

One of the first complications to address, having in mind the general streaming pipeline (shown in Fig. 1.5) is the diversity of data sources (json, binary, xml, avro, ...) and storage systems (RDBMS, S3, NoSql, Kafka, ...) that could come into play in a complex system. For this reason, you would need technologies that can manage them in a standard and common way in order to avoid compatibility problems between the code and the different environments.[9]

Fig. 1.5: General streaming pipeline

Secondly, you should think about which type of operations to apply to our data and consequently which type of output is needed.

Stateful operations Functions that aggregate or count data come under this scenario, where an incoming record depends upon the result of previously processed records.

Therefore, you need to maintain an intermediate information, called **state of data**, that every record may read and update.[10]

Stateless operations Functions like *map()* and *filter()* come under this scenario, in which every incoming record is independent of other records and can be, for this reason, processed and persisted independently. Here, the streaming process does not have a state of data, simply because you do not need one.[10]

1.3.2 System reliability

There is another important information that is the bare minimum for any streaming system and it is expected in both stateless and stateful processing: the **state of progress**. It means keeping track of data that has been processed so far in order to have reliable results. This is connected with a key characteristic any architecture should have: the fault tolerance in case of events such as restart, upgrade or task failures. In this case, the stored state of progress is a fundamental information because you have to ensure the atomicity of what is processed.[10]

1.3.3 Batch size and windowing

As you understood so far, you can never really hope to get a global view of a data stream. Hence, even if you are in a streaming setting, you must somehow partition the data into micro-batches in order to analyse it.

Moreover, speaking about the state of data, it is not optimal to store it neither indefinitely nor for the entire session, because over time it could grow a lot. Indeed, it is reasonable to preserve that state the least time possible.

This gives the rise to two important concepts:

- **micro-batch size**: the number of records belonging to a single event to analyse;
- **window length**: the number of micro-batches for which you need to keep stored intermediate results before deleting them. It might be 5 minutes, 2 hours, or maybe the last 256 events; it is completely use-case-dependent.

There is a number of ways in which you can window data and later process the results.

Different technologies handle this concept with different approaches: we will try now to do a general overview, without diving into a specific implementation.

Time During processing, each data element in the stream needs to be associated with a timestamp in order to create the windows; this can be done in three ways:

- event-time: it is data-dependent and embedded in the event itself;
- ingestion-time: it is assigned to the event when it enters the system;
- processing-time: the wall-clock time when the event is processed.

Event-time, while most useful, is also the most troubling: data may arrive out of order or late, so we can never be sure if we saw all events in a given time window. Processing-time is less useful but it is the easiest, as it is monotonic: you know precisely when a window ended by looking at the clock.

Types of windows As shown in Fig. 1.6, windows can be of different types.

- Fixed/tumbling windows are equal and non-overlapping chunks; each event belongs to exactly one of them.
- Sliding windows have fixed length too, but they are separated by a time interval, called sliding interval. The window length is usually a multiplicity of the step: in this way each micro-batch belongs to a number of windows without being divided.
- Session windows have various sizes and they are defined basing on data, which should carry some session identifiers.

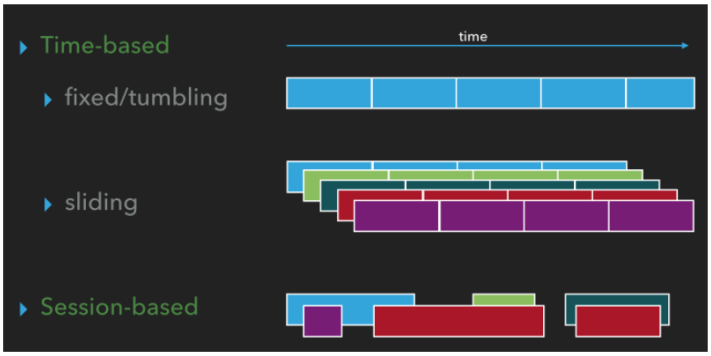

Fig. 1.6: Types of windows

Out-of-order data As hinted before, data can arrive out-of-order. For example, when receiving a stream of sensor readings, devices might be offline and send catch-up data after some time. That is why streaming systems definitely have to allow for some lateness in event arrival; the problem is how much. At some point, a window has to be considered "done" and garbage collected.

Some technologies handle this issue with a default window closing-time, others instead use a mechanism called **watermarks**. Basically, a watermark is a threshold to specify how long the system waits for late events. If an arriving event lies within the watermark, it gets used to update the result of the specific window, otherwise it will be dropped and not further processed by the system.

Triggers Lastly, having a way to obtain a value from a window, you still need to decide when to run the computation. This concept is called **trigger** and in the following there is the list of some possibilities:

- watermark progress: compute the final window result once the watermark passes the window boundary;
- event-time progress: compute window results multiple times, with a specified step, as the watermark progresses;
- processing-time progress: compute window results multiple times based on a given interval measured against wall-clock time. [11]

2 Different approaches

2.1 Data Warehouse and Data Lake

2.2 Lambda architecture vs Kappa architecture

2.3 Beyond Lambda: Lake House

2.4 (Data virtualization)

3 Spark engine

3.1 What is Apache Spark?

3.2 RDD vs Dataframes/Datasets

3.3 Spark Streaming vs Spark Structured Streaming

3.4 Challenges of Streaming Data

3.4.1 Checkpoints

3.4.2 Windows

3.4.3 Watermarks

4 Use case

4.1 Platform overview

4.2 Problem statement

4.3 Proposed solution

4.3.1 Other technologies used

4.3.2 Data gathering (which dataset , etc...)

4.3.3 Data transformation (logics behind the scene)

4.3.4 Data storage (On-prem vs Cloud -> chapter 5)

4.3.5 Data visualization (Power BI, web applications, etc...)

4.4 Results

5 On-prem vs Cloud

5.1 Main theoretical differences

5.2 Performance measures: costs, scalability, fault-tolerance, privacy, ...

5.3 Experimental evaluation

6 Machine learning with Streaming Data

6.1 ...

Conclusion

References

[1] Oracle. *What is Big Data?* URL: https://www.oracle.com/it/big-data/what-is-big-data/.

[2] Douglas Laney. *3D Data Management: Controlling Data Volume, Velocity, and Variety*. Tech. rep. META Group, Feb. 2001. URL: http://blogs.gartner.com/doug-laney/files/2012/01/ad949-3D-Data-Management-Controlling-Data-Volume-Velocity-and-Variety.pdf.

[3] Andrea Brunello. "Introduction to Big Data". Trieste, Italy, 2020. URL: https://github.com/dslab-uniud/teaching.

[4] Malika Harkati. *What Is Big Data and How Does It Work?* CLOUDit-eg. URL: https://cloudit-eg.com/what-is-big-data-and-how-does-it-work/.

[5] Gartner Glossary. *Real-time Analytics*. Gartner. URL: https://www.gartner.com/en/information-technology/glossary/real-time-analytics.

[6] Christopher Tozzi. *Dummy's Guide to Batch vs. Streaming Data*. Precisely. URL: https://www.precisely.com/blog/big-data/big-data-101-batch-stream-processing.

[7] Mark Balkenende. *The Big Data Debate: Batch Versus Stream Processing*. The New Stack. URL: https://thenewstack.io/the-big-data-debate-batch-processing-vs-streaming-processing/.

[8] Databricks. *What is Streaming Analytics?* URL: https://databricks.com/glossary/streaming-analytics.

[9] Tathagata Das. *Easy, Scalable, Fault-Tolerant Stream Processing with Structured Streaming in Apache Spark*. Databricks. URL: https://databricks.com/session/easy-scalable-fault-tolerant-stream-processing-with-structured-streaming-in-apache-spark-continues.

[10] Hadoop For Everyone. *Spark Streaming : Stateless Vs Stateful operations Explained*. URL: https://www.youtube.com/watch?app=desktop&v=0C9MHMYyCYY&feature=youtu.be.

[11] Adam Warski. *Windowing data in Big Data Streams*. Softwaremill. URL: https://softwaremill.com/windowing-in-big-data-streams-spark-flink-kafka-akka/.

Appendix

A1 Test

Fig. A1.1: NHH logo