THE IMPORTANCE OF BIG DATA IN GAZ-CHEMICAL INDUSTRIES

F.A.Adilov
Prof.
M.M.Xomidov
student

Abstract: *Nowadays, most Gaz-chemical industries including UzGTL have been focusing on enhancing their productivities and avoiding problems which may occur while running manufacturing process. To achieve this, main address is about to produce BIG DATA based on their potential afford. As a BIG DATA provides opportunities to making fast and right decision in real time and even for future forecasts based on all data stream coming from different sectors of an industrial company. When the Big data platform is developed by Phyton or MATLAB programming language, it becomes more effective than traditional ones as these programming languages are highly to build strong connection between software and hardware of the industrial sectors.*

Keywords: *Phyton, future forecast, Hadoop, Spark, Machine learning, decision making, data transformation, extracting and analyzing information.*

INTRODUCTION

The evolution of the technologies in Big Data in the last 20 years has presented a history of battles with growing data volume. The challenge of big data has not been solved yet, and the effort will certainly continue, with the data volume continuing to grow in the coming years. The original relational database system (RDBMS) and the associated OLTP (Online Transaction Processing) make it so easy to work with data using SQL in all aspects, as long as the data size is small enough to manage. However, when the data reach a significant volume, it becomes very difficult to work with because it would take a long time, or sometimes even be impossible, to read, write, and process successfully. Overall, dealing with a large amount of data is a universal problem for data engineers and data scientists. The problem has manifested in many new technologies (Hadoop, NoSQL database, Spark, etc.) that have bloomed in the last decade, and this trend will continue. This article is dedicated on the main principles to keep in mind when you design and implement a data-intensive process of large data volume, which could be a data preparation for your machine learning

applications, or pulling data from multiple sources and generating reports or dashboards for your customers.

The essential problem of dealing with big data is, in fact, a resource issue. Because the larger the volume of the data, the more the resources required, in terms of memory, processors, and disks.

THE METHOD OF EXTRACTING INFORMATION COMING FROM SECTORS OF INDUSTRILIZED GAZ-CHEMICAL COMPANY AND ANALYIZING STORED DATA WITH PROGRAMMED CONTROLLING SYSTEMS.

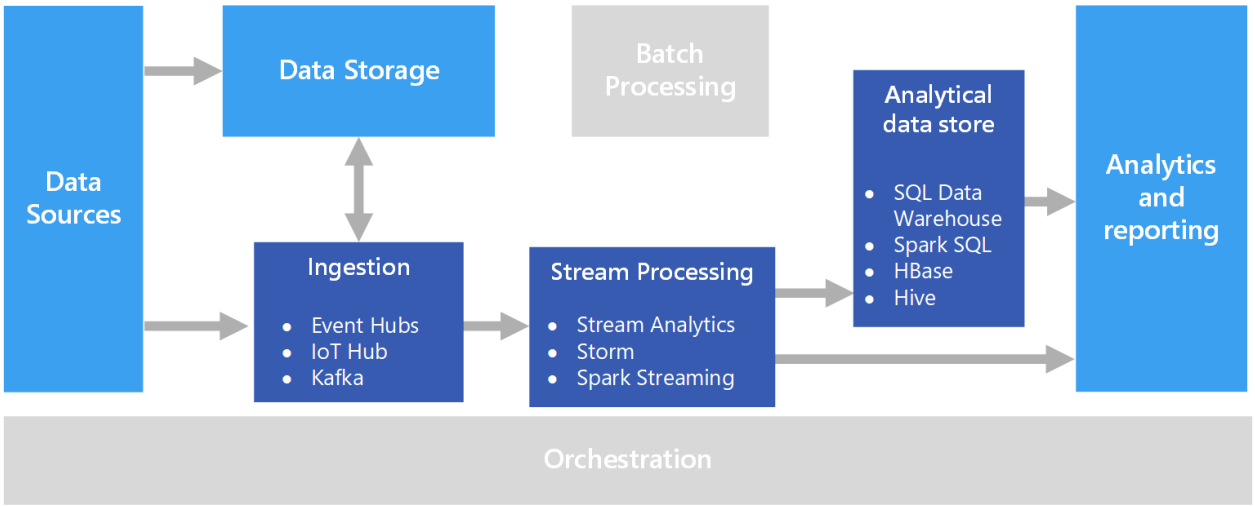
The goal of performance optimization is to either reduce resource usage or make it more efficient to fully utilize the available resources, so that it takes less time to read, write, or process the data. The ultimate objectives of any optimization should include:

- Maximized usage of memory that is available
- Reduced disk I/O
- Minimized data transfer over the network
- Parallel processing to fully leverage multi-processors

Therefore, when working on big data performance, a good architect is not only a programmer, but also possess good knowledge of server architecture and database systems. With these objectives in mind, let's look at 4 key principles for designing or optimizing your data processes or applications, no matter which tool, programming language, or used framework.

1. Design based on your data volume.
2. Reduce data volume earlier in the process.
3. Partition the data properly based on processing logic.
4. Avoid unnecessary resource-expensive processing steps whenever possible.

Real time processing deals with streams of data that are captured in real-time and processed with minimal latency to generate real-time (or near-real-time) reports or automated responses. For example, a real-time traffic monitoring solution might use sensor data to detect high traffic volumes. This data could be used to dynamically update a map to show congestion, or automatically initiate high-occupancy lanes or other traffic management systems.



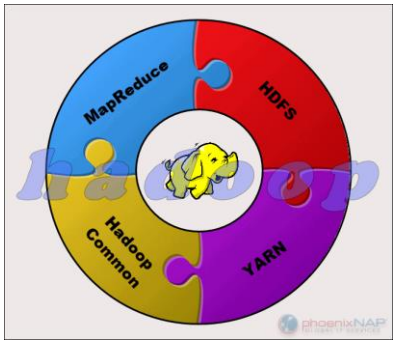
Real-time processing is defined as the processing of unbounded stream of input data, with very short latency requirements for processing — measured in milliseconds or seconds. This incoming data typically arrives in an unstructured or semi-structured format, such as JSON, and has the same processing requirements as batch processing, but with shorter turnaround times to support real-time consumption.

Processed data is often written to an analytical data store, which is optimized for analytics and visualization. The processed data can also be ingested directly into the analytics and reporting layer for analysis, business intelligence, and real-time dashboard visualization.

In terms of interacting with big data, different programming languages will accomplish different sectors when working with the large stream of data. There are some including Phyton, Java, Scala (a Java based-language).

Python: Python is an open-source language, and is regarded as one of the simplest to learn. It utilizes concise syntax and abstraction. Because of its popularity and open-source nature, it has an extensive community of support with near-endless libraries that enable scalability and connections with online applications. It is compatible with Hadoop.

Apache Hadoop is a platform that handles large datasets in a distributed fashion. The framework uses **MapReduce** to split the data into blocks and assign the chunks to nodes across a cluster. MapReduce then processes the data in parallel on each node to produce a unique output. Every machine in a cluster both stores and processes data. Hadoop stores the data to disks using **HDFS**. The software offers seamless scalability options. You can start with as low as one machine and then expand to thousands, adding any type of enterprise or commodity hardware. The Hadoop ecosystem is highly fault-tolerant. Hadoop does not depend on hardware to achieve high availability. At its core, Hadoop is built to look for failures at the application layer. By replicating data across a cluster, when a piece of hardware fails, the framework can build the missing parts from another location.



The Apache Hadoop Project consists of four main modules:

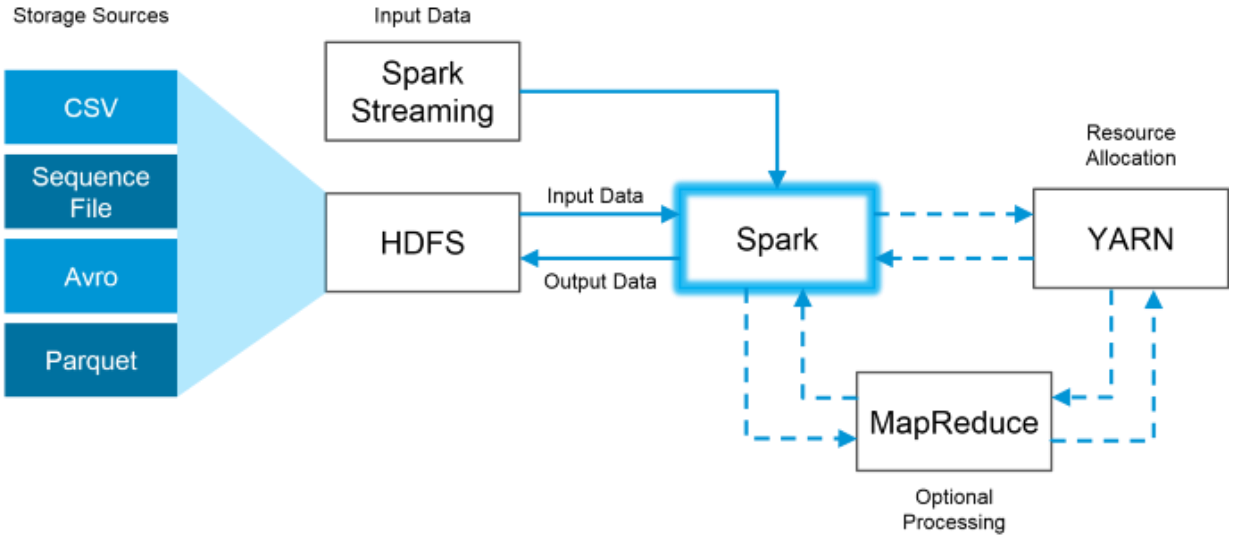
1. **HDFS** – Hadoop Distributed File System. This is the file system that manages the storage of large sets of data across a Hadoop cluster. HDFS can handle both structured and unstructured data. The storage hardware can range from any consumer-grade HDDs to enterprise drives.

2. **MapReduce.** The processing component of the Hadoop ecosystem. It assigns the data fragments from the HDFS to separate map tasks in the cluster. MapReduce processes the chunks in parallel to combine the pieces into the desired result.

3. **YARN.** Yet Another Resource Negotiator. Responsible for managing computing resources and job scheduling.

4. **Hadoop Common.** The set of common libraries and utilities that other modules depend on. Another name for this module is Hadoop core, as it provides support for all other Hadoop components.

The main reason for this supremacy of Spark is that it does not read and write intermediate data to disks but uses RAM. Hadoop stores data on many different sources and then process the data in batches using MapReduce.



The great news is that Spark is fully compatible with the Hadoop eco-system and works smoothly with ***Hadoop Distributed File System (HDFS)***, Apache Hive, and others. So, when the size of the data is too big for Spark to handle in memory, Hadoop can help overcome that hurdle via its HDFS functionality. Below is a visual example of how Spark and Hadoop can work together:

Method

In this section the two proposals are fully described. Even though the algorithms have been run on both Spark and Hadoop, the explanation is in common since the philosophy is the same and the unique difference is the platform. Then, In the experimental stage, these two proposals have been compared to existing AC approaches to demonstrate that both are the most interesting ones. The experimental set-up is also fully described in this section including which comparisons have been performed. Generation of association rules. The goal of this phase is to obtain those rules whose frequency of occurrence is greater than a threshold value predefined by the user. In this sense, an iterative algorithm based on the well-known Apriori [2] is considered. An important problem of this kind of methodologies is the extremely high number of rules that may be produced at the same time. To deal with this issue, an option is not to create the whole lattice for each transaction but the l-sized sub-lattice each time, requiring a predefined number of iterations. In this regard, both the memory requirements and the computational time can be reduced. Furthermore, when the l-sized rules are generated, only the supersets from l-1-sized frequent rules are used as a seed, enabling to speed up the runtime. The explanation behind this fact is that any rule can

be only frequent if all its sub-rules are also frequent [2]. This phase has been developed with a classical MapReduce application including three different types of processes: 1) driver: it is the main program when running the algorithm; 2) mappers: they aim at processing an input and producing a set of k, v pairs; 3) reducers: they receive the previously set of pairs in order to aggregate and filter the final results. Each of these steps are fully described as follows:

● Step 1. The driver reads the database from disk and save it in the main memory of the cluster. Either using Spark or Flink, the driver splits the dataset in data subsets in order to ease both the data access and data storage.

● Step 2. Each time this phase (see Listing 1) is performed a MapReduce procedure is run. The proposed model works by running a different mapper for each specific sub-database and, then, the results are collected and filtered in a reducer phase. These two sub-steps are described as follows.

● Step 3. After the MapReduce phase is carried out, the best rules of size l are reported to the driver.

Listing 1 CBA-Spark/Flink - Generation of association rules - Step 2

```
function mapper(instance, l, l-1-sizedRules)
1: candidates ← generateRulesSizeL(l, instance)
2: for all candidate in candidates do
3:     if candidate is not a subset of any l-1-sizedRules then
4:         supports ← calculateSupports(candidate, instance)
5:         emit(candidate, supports) // Emit ⟨k, v⟩ pair
6:     end if
7: end for
end function
function reducer(candidate, supports)
1: finalSupports ← {supportAntecedent: 0, supportConsequent: 0, supportRule: 0}
2: for all support in supports do
3:     finalSupports ← finallSupports + support
4: end for
5: if finalSupports.supportRule ≥ threshold then
6:     emit(candidate, supports) // Emit ⟨k, v⟩ pair
7: end if
end function
```

CONCLUSION

The importance of producing big data has been the global need among all industries as it gives advantages to build up stronger connection between all sectors and having more qualified results in real time. Actually, Hadoop and Spark mentioned above have the most important role to set up big data platform in industrialized sectors including transforming gas to liquid. As technology enhances, the need to gain more detailed information will increase. Thus, gas-chemical industries should be adopted as soon as it can be in order to enhance productivity and minimize risks. Programming local communication of all devices in Phyton to work in one main PC is the important task that most researchers and developers are now focusing on.

LITERATURE:

1. How to choose Hadoop or spark in building big data. In 2021 https://www.quora.com/What-is-the-difference-between-Hadoop-and-Spark

2. Hadoop components in big data platform. Programming process.2020. https://www.quora.com/Is-Spark-a-component-of-the-Hadoop-ecosystem

3. Agrawal R, Imielinski T, Swami A. Mining association rules between sets of items in large databases. SIGMOD Rec. ´ 1993;22(2):207–16.

4. Lam C. Hadoop in Action, 1st edn. Greenwich, CT, USA: Manning Publications Co.; 2010.

5. Evaluating associative classification algorithms for Big Data. Francisco Padillo1, José María Luna1,3 and Sebastián Ventura1,2,3*

6. 2. Venturini L, Baralis E, Garza P. Scaling associative classification for very large datasets. J Big Data. 2017;4(1):44. https://doi.org/10.1186/s40537-017-0107-2. 33.

7. del Río S, López V, Benítez JM, Herrera F. On the use of mapreduce for imbalanced big data using random forest. Inf Sci. 2014;285:112–37