**BİL401**

**Final Report**

**Kerem Elma - 211101079**

**Mehmet Eski - 221104095**

**TABLE OF CONTENT**

1. **Introduction**
2. **Codebase and Materials**
3. **Platform Details**
4. **Results**
   1. **Read Rate and Memory Usage**
   2. **Write Rate and Memory Usage**
   3. **File Size**
   4. **Relocation Time**
   5. **Disk I/O Performance**
   6. **Compression Ratio**
5. **Introduction**

This report compares of Avro, Parquet, XML, JSON data formats under the concept of big data. In the comparisons, different metrics were tried and it was tried to determine which one is more optimal to use. With the help of various technologies such as PySpark, Hadoop, Apache Hive; the scope has been expanded and metrics have been diversified.

1. **Codebase and Materials**

The first version of the data was in JSON format. with a file converter python script we wrote ourselves, we created the same data in other formats and added it to our datasets. In these Python scripts, libraries such as pandas, pyarrow, json was used which help in conversions.

In the same way, with Spark's PythonAPI, called PySpark, we wrote reader scripts to read these files, these scripts were responsible for reading the dataset on the local machine.

We used Hadoop’s HDFS for the storage part and again observed the processing of this big data for various metrics.

We also used Apache Hive to analyze this data using SQL-like queries.

The complete codebsae is available at the following github link. In the main path, the **file\_converter.py** file converts the original json dataset into different formats. In the **datasets/** path, there are the files created by the converter. In the path named **readers/**, there are pyspark scripts that read the files in these formats.

* <https://github.com/kelma01/dataComparer>

1. **Platform Details**

For the implementation, we used Python language and pyspark provided by spark. We chose Python because it is easy to write, flexible and Spark provides such a possibility.

For storage, we chose Hadoop’s HDFS because it is quite common and convenient to use for storage and management in big data processing environments. Its distributed processing is suitable for parallel processing. It is also important that it is compatible with the data formats we will be using. Since we are using Hadoop, we used a cluster infrastructure. Hadoop's big data processing power comes from its ability to perform parallel processing on multiple machines (nodes).

For data management, we chose Apache Hive, a SQL-like query language used for big data analytics on Hadoop. It stores data on HDFS and performs fast analysis with distributed processing power. Hive is a tool built on top of the Hadoop cluster. Therefore, because of Hive, we used the distributed computing infrastructure in the Hadoop cluster in the background.

1. **Results**

The results obtained with different metrics for data processed in different formats are given in the following sections:

* **Read Rate and Memory Usage:**

The following outputs show how many seconds it took to read the file and the memory usage during that time on a local machine without any storage or database:

metin, ekran görüntüsü, yazı tipi, çizgi içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir. ekran görüntüsü, metin, yazı tipi, çizgi içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir. metin, ekran görüntüsü, yazı tipi, çizgi içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir. metin, ekran görüntüsü, yazı tipi, çizgi içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

When we look at the outputs, it is seen that the performance of Avro and Parquet is much superior to the JSON and XML format. This is because Avro and Parquet are binary formatted and schema based. At the same time, reading times are much shorter due to their compressed format. Therefore, when looking at the read speed metric, using Avro and Parquet can be seen as an optimal solution. In datasets, as the size is small, the read times of different formats are quite close to each other, but as the size difference increases, this read speed difference increases for text-based formats such as XML and JSON.

Parquet has the lowest memory usage of the four. It optimizes disk read time because it reads schema-based and related columns. Low memory consumption can be advantageous when working with large data sets. When we look at the others, Avro format also uses as much memory as JSON and

XML, but still has a lower read time than them because, like Parquet, it is schema-based and compressible, which makes it stand out in this field.

As we mentioned, we used Hadoop's HDFS for storage. With HDFS, processing, storing and managing big data has become much easier. The outputs below show how many seconds it takes to read a file through an HDFS server and the memory usage during that time:  
  
metin, ekran görüntüsü, yazı tipi, çizgi içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir. metin, ekran görüntüsü, yazı tipi, çizgi içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir. metin, ekran görüntüsü, yazı tipi, çizgi içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir. metin, ekran görüntüsü, yazı tipi, çizgi içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

Comparing reading from local disk with reading from HDFS, reading from local disk is generally faster, but consumes more memory. For JSON and XML formats, reading time from HDFS is significantly longer, with XML in particular showing a significant performance degradation. Parquet and Avro formats, show relatively stable performance on HDFS, but read times are still longer than on local disk. In terms of memory usage, the memory consumption of the JSON reading from HDFS is significantly lower compared to local disk, which can be explained by the stream-based data processing strategy of HDFS. In particular, the Parquet format has the lowest memory consumption both locally and on HDFS, which is due to the advantages of column-based compression and data storage.

In general, although read operations from HDFS are slow due to network latency and block management, it offers a more efficient approach in terms of memory management.

* **Write Rate and Memory Usage**

This section tests appending the same data to different formats. The following terminal output shows the time and memory usage during the test:

metin, ekran görüntüsü, yazı tipi, çizgi içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.metin, ekran görüntüsü, yazı tipi, çizgi içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.metin, ekran görüntüsü, yazı tipi, çizgi içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.metin, ekran görüntüsü, çizgi, yazı tipi içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

In terms of write rate and memory usage, XML is the slowest and most memory consuming format. This is an expected result due to XML's extensible nature and the need for extra tagging. While the JSON format is close to XML in terms of write time, it stands out with much lower memory usage. Parquet format stands out as the most efficient option in terms of both write time and memory usage. Due to its column-based format and compression support, it is especially advantageous in big data processing. Avro, on the other hand, has a faster write time compared to JSON and has a balanced performance in terms of memory usage.

Overall, Parquet offers the best performance results, while Avro offers a good alternative in terms of speed and efficiency.

* **File Size:**

After the files are created by the file converter, they keep the same data content in just different formats. After this conversion, the sizes of the files are as follows:

metin, ekran görüntüsü, yazı tipi içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

The numbers written next to the files are the space the files hold in memory in KB. If we interpret this output, the format that can be stored in the smallest size is Parquet, while the format that can be stored in the largest size is XML.

If we interpret this output, the format that can be stored in the smallest size is Parquet, while the format that can be stored in the largest size is XML.

The reason why Parquet and Avro are the best formats here is because of their compressibility feature, which we mentioned before. This provides advantages in storage and reading performance.

JSON and XML, on the other hand, can take up extra space because they are text-based and because of the key-value structure in JSON and the tag structure in XML.

We used Hadoop's commands to move the datasets created by the file converter to HDFS. After the conversion process, files are moved into HDFS with the `hadoop fs -put pathOfLocalFiles hdfsPathOfFiles` command.

After processes, the sizes of the files are as follows:  
  
metin, ekran görüntüsü, yazı tipi içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

In the results we get from the Hadoop’s output, the sizes are shown in byte format. When we convert these byte values to kilobytes, the sizes of the files on local, as seen in the previous header, become comparable since they are also in kilobytes.

Comparing the file sizes on local storage and HDFS, it is clear that the files on HDFS are larger than those on the local storage. In particular, JSON and XML formats take up much more space on HDFS, which may be due to the block based storage structure of HDFS and the metadata situation. Parquet format has the smallest file size in both environments, confirming its advantages of compression and column based storage. Avro format, takes up significantly less space than JSON and XML, which is due to the fact that it is a schema and binary formatted.

Overall, it can be seen that HDFS, although optimized for big data management, consumes more storage space than local. This can be explained by the fact that HDFS has additional metadata and block management mechanisms to ensure data integrity and fault tolerance.

In addition, in the sizes of files are presented in two columns, size in the first column represents the raw size of the data, the actual value. The byte value in the second column represents the total storage usage on HDFS, including replication factors. The fact that the values in these two columns are equal in the output also tells us that this replication factor is 1.

* **Parallelism, Executor and Core usage:**

The following outputs represent the performance analysis of this different data formats in terms of parallelism, executor instances, core utilization, memory usage and on Apache Spark:

metin, ekran görüntüsü, yazı tipi, çizgi içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.metin, ekran görüntüsü, yazı tipi, yazılım içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.metin, ekran görüntüsü, yazı tipi içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.metin, ekran görüntüsü, yazı tipi içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

The number of partitions allows for faster processing by breaking the data into smaller chunks. Formats such as Parquet and Avro can work just as efficiently with fewer partitions because they have an optimized structure, but relatively irregular formats such as JSON and XML require more processing power to parse and process the data directly and this means more partition operations.

Number of executors is intended to increase processing speed by enabling threads to run concurrently. They must be used in conjunction with the partition number, if the number of executors are higher than the partition, this can lead to idle executors. As mentioned earlier, in JSON and XML formats, executors are used effectively because the partition is used more.

The number of cores usually comes from the parallelism value that comes by default. Number of cores determines the number of tasks per executor, of course. This tells us that more cores can do more concurrent work.

When we run the same tests with the datasets we read through HDFS and not with the dataset available on local storage, we see that there is no significant difference, but a real difference can be seen when a larger dataset is used.

If these tests were done in a real distributed environment, HDFS would have an advantage due to its data locality feature. At the same time, since the test is running in standalone mode on a single machine, we may not be able to see these advantages of HDFS.

This data locality provided by HDFS is the mechanism of storing the data in the location closest to the computation. In this way, instead of moving the data to the processor, processing it there and then storing it back, it does the processing where the data is. In this way, big data operations become faster and more efficient.

* **Disk I/O Performances:**

To measure the I/O operations performed on disk while moving data from local storage to HDFS, we used a command on the computer that tracks and reports the operations performed on disk. After starting the command and then relocating the file, we analyzed the file it provides as output.

Parquet and Avro, which are generally more efficient formats for big data, are advantageous for read operations with features such as column-based and schema-based structures. In text-based formats such as XML and JSON, big data operations require more disk I/O operations. However, since the size of the datasets used in the tests was not large enough, the I/O operations were not very noticeable, but the differences between the formats can be clearly seen when done with larger datasets in the real world.

In such a scenario, Avro and Parquet will offer the same efficiency in read/write speed on disk as they usually do.

* **Relocation Time:**

When we calculate the time it takes to execute the `hadoop fs -put pathOfLocalFiles hdfsPathOfFiles` command, we see that there is a relationship between the file size and the duration of the put operation, and this relationship is directly proportional. A file with a larger format size than other formats, for example XML, takes longer to be moved to HDFS.

* **Compression Ratio**

As mentioned in the previous sections on size, it is understood that Parquet and Avro have a compression feature. This compression feature allows for faster reading operations with a smaller footprint.