PERFORMANCE STUDY OF PARQUET CODECs

AUGUST 2019

AUTHOR(S):
Javier García Rubio

CERN IT-DB Group

SUPERVISOR(S):
Manuel Martin

CERN openlab

PROJECT SPECIFICATION

Apache Parquet is a columnar storage format for the Hadoop ecosystem. The technology has become almost a de-factor standard due to important benefits and adventages such its seameless integration with multiple choices of data processing framework, data model or programming language.

Another important aspect of this technology is the capacity to redure drastically the amount of storage required to persist data. That reduction is achieved based on the columnar format used but also to the compression codecs supported and applied to the data once it is transformed into parquet format.

This project studies the implications in terms of performance on data compression and access of the differents codecs available.

ABSTRACT

This report describes the work carried out to study and evaluate the performance and footprint of different parquet compression codecs on data retrival and analytics scenarios Parquet is a standard-de-facto and the data format used to persist and analyse mission-critical data coming from CERN's accelerators and experiments control systems.

The study applied Hadoop related technologies such as Scala and Spark and is based on real data obtained from the Next Generation Accelerator Logging Service (NXCALS). This data is compressed with the different codecs evaluated and after that we compare different relevant metrics like the compression rate and time AVG.Finally the results are evaluated and several conclusions are exposed.

.

TABLE OF CONTENTS

ACKNOWLEDGMENTS

I would like to thanks to my supervisor Manuel Martin for helping me with this project, for trusting on me and making me feel comfortable since the first day. I also would like express my special gratitude to CERN in general and CERN openlab team for giving me this opportunity.

1. INTRODUCTION

The objective of this project is to evaluate the performance of the different codecs from Parquet to have the best result, considering the amount of data at CERN, which use Parquet. In this project we use real data from NXCALS (New Generation Accelerator Logging Service).

We studied how the different codecs (compression) affect the storage and access to the data. Data Access Performance, the CPU payload are evaluated and different configurations are tested in order to reduce those factors and get result for later comparisons among them and with new codec that Oracle is currently developing.

2. TECHNOLOGIES

In this project we applied Hadoop technology stack with special focus on Spark and Scala to create to develop the necessary performance testing tools and scripts. In addition, some special Hadoop and Java libraries are also used..

The decision toward this set of technologies was a natural choice based on the specifics of the project, that advantages of Hadoop, Java support and community for Spark and Scala.

```
import java.io._
import java.lang.management.ManagementFactory
import scala.concurrent.duration._
import scala.sys.process._
import org.apache.hadoop.fs.FileSystem
import org.apache.hadoop.fs.Path
import org.apache.spark.scheduler._
```

Figure 1. Libraries that I use

3. WORKFLOW

a. EXPERIMENTAL SETUP

For all the test performance we used a cloud based cluster (CERN openstack private cloud) with Hadoop environment, we used Hadoop version 3.1.1.

On that environment is so optimal use Spark and Scala to create the script, both are supporting with a big community, their base is on Java and they work with HDFS (Hadoop Distributed File System) which is an important data for that project.

We use Spark version 2.4.3 and version 2.13 for Scala (they are the most recent version on that moment).

b. METHODOLOGY

In order to obtain a representative subset of data for the study, instead of obtaining data from a single system we focused on a time window. All data generated by the control system on a specific and limited time window is used, the time window was carefully selected taking into account factors such as the type of physic performed and the operation status of the accelerator at that time. Once that time window was identified, we needed to extract the corresponding data and move it from the production hadoop cluster to another hadoop cluster in order to ensure both that our benchmarks do not affect the production cluster and also that our results are not biased by potential and external workload on the cluster.

```
hadoop fs -copyToLocal hadoop_path local_path
```

Figure 2. Command to copy data from Hadoop cluster to local

Once the data is in our target test cluster, we need to obtain the list of files and use them as an input for the benchmarks

```
hadoop fs -find directory_name | grep parquet >> test
```

Figure 3. Command to get parquet files and write it on a test file

The benchmark script will take the file paths obtained in the previous step and compress them using the different codecs for parquet to obtain the performance metrics. These codecs are LZ4, GZIP, SNAPPY and Parquet (No codec). The transformation workflow applied can be seen in figure 3 and 4.

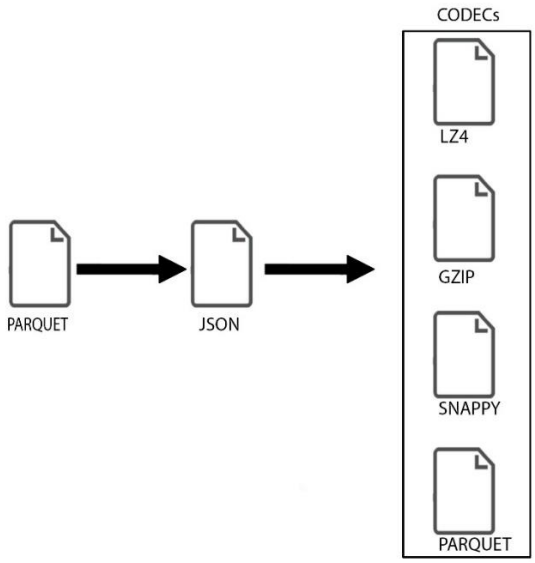

Figure 4. Parquet to different codecs

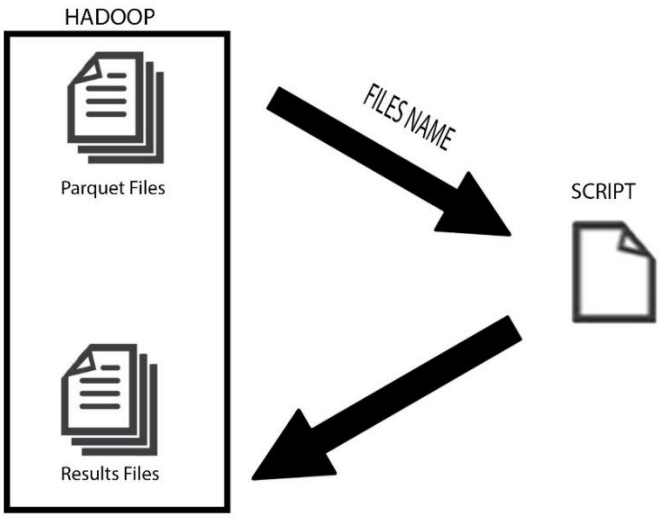

Figure 5. The workflow of the project

4. PERFORMANCE ANALYSIS

a. INTRODUCTION

We save the metrics while the script was running on a CSV file with the filename, format, size (KB), size (MB) and time (ms). We used that metrics for the next step, evaluated the CPU time average and the compression rate.

Filename	Format	Size (KB)	Size(MB)	Time (ms)
00_24_H-part-00000-e426d360-90ab-4128-9a6f-100a1775d86c-c000.json	JSON	385951KB	376MB	2009
00_24_H-part-00000-e426d360-90ab-4128-9a6f-100a1775d86c-c000.gzip.parquet	GZIP	12076KB	11MB	3905
00_24_H-part-00000-e426d360-90ab-4128-9a6f-100a1775d86c-c000.lz4.parquet	LZ4	14405KB	14MB	3121
00_24_H-part-00000-e426d360-90ab-4128-9a6f-100a1775d86c-c000.snappy.parquet	SNAPPY	14724KB	14MB	3039
00_24_H-part-00000-e426d360-90ab-4128-9a6f-100a1775d86c-c000.parquet	PARQUET	14724KB	14MB	3101

Figure 6. CSV File

b. COMPRESSION RATE

We take the json file size and then we compare with the size of the appropriate codec compression, and we take the rate for each file, in that case the best codec is GZIP with a little bit advantage (0,8% and 0,9%). The size from the real files that we used is variable, they are between 17KB to 70GB.

CODEC	% COMPRESSION
GZIP	96,8%
LZ4	96%
SNAPPY	95,9%
PARQUET	95,9%

Figure 7. Compression rate table

c. CPU TIME

We get the CPU time (ms) just when the script starts the compression. The fastest is LZ4, and GZIP, which is the one with the best compression rate, now we can observe that is the slowest one, and snappy, which have the same compression rate as parquet, now we see that is faster and it is almost equal with LZ4.

CODEC	TIME AVG (ms)
GZIP	7784,5
PARQUET	7157,5
SNAPPY	6921
LZ4	6867

Figure 8. Time AVG table

5. CONCLUSIONS

The results show no significant differences among the different codec evaluated. However, LZ4 has overcome in some aspects the other codecs. There isn´t too much difference between Snappy and LZ4, the compression and time percentages difference with LZ4 is 0,1% and we can appreciate the difference with GZIP that it compresses 0,8% more but is 12% slower. The test was done with about 3000 real files with sizes ranging from a few KB to GB from NXCALS. Based on the result we can conclude that despite of the small difference on performance perceived the use of LZ4 instead of snappy (currently used in production for NXCALS) can improve the current performance.

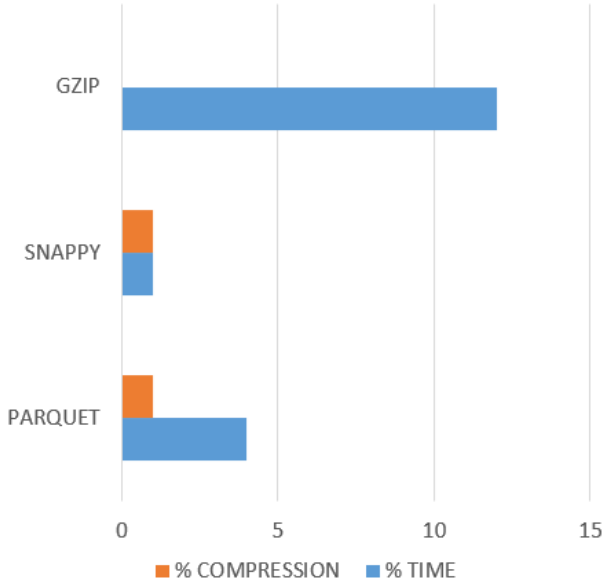

Figure 9. Compare LZ4 with the other codecs

6. NEXT STEPS

The project was born to serve as a basis for a more advance performance study of the existing codes on regards to a new codec approach that is currently being developed by Oracle, Parse Aware Compression (PAC). Therefore, the next steps are to extend this study with the new codec and determine whether the new approach introduce significant differences and its understand advantages and disadvantages.

7. REFERENCES

-Spark Documentation (https://spark.apache.org/docs/latest/index.html)

-Scala Documentation (https://docs.scala-lang.org/)

-Hadoop Documentation (https://hadoop.apache.org/docs/r2.4.1/hadoop-project-dist/hadoop-hdfs/HdfsUserGuide.html)