A Comparison of the Efficiencies of Various Structured and Semi-Structured Data Formats in Data Analysis and Big Data Analytic Development

Heather E. Graham and Taoxin Peng
School of Computing, Engineering and the Build Environment, Edinburgh Napier University, Edinburgh, U.K.

Keywords: Athena, Cloud, Big Data, Data Format, Semi-Structured, Efficiency, EMR.

Abstract: As data volumes grow, so too does our need and ability to analyse it. Cloud computing technologies offer a wide variety of options for analysing big data and make this ability available to anyone. However, the monetary implications for doing this in an inefficient fashion could surprise those who may be used to an on-premises solution to big data analysis, as they move from a model where storage is limited and processing power has little cost implications, to a model where storage is cheap but compute is expensive. This paper investigates the efficiencies gained or lost by using each of five data formats, CSV, JSON, Parquet, ORC and Avro, on Amazon Athena, which uses SQL as a query language over data at rest in Amazon S3, and on Amazon EMR, using the Pig language over a distributed Hadoop architecture. Experiment results suggest that ORC is the most efficient data format to use on the platforms tested against, based on time and monetary costs.

1 INTRODUCTION

Data is growing in size and in variety at an ever-increasing pace, as are the technologies that are used to analyse it. The combination of analysis, data formats and tooling used has the potential to be very powerful for businesses, but the wrong combination also has the potential to be very costly.

Once, only a few businesses had the profits and ability to be able to analyse ‘big’ data with databases or data warehouses they managed and maintained themselves. The advent of cloud computing has made this accessible to everyone. In an ecosystem where storage is considered cheap whereas computational power is expensive, data can be stored in multiple data formats to allow analysis to be conducted efficiently and with less expense.

For businesses who are moving towards cloud computing from previous on-premises big data solutions, this is a different mindset, as previously they would have been restricted by storage size and the monetary value of compute power was subsumed in the initial setup costs of their big data architecture.

Different data formats have been designed with different functions in mind, with row-oriented formats being designed to efficiently run over subsets of rows in data, that might be defined using a WHERE or FILTER function, whereas columnar formats are designed to run efficiently over a subset of columns within the data. For this reason, it is important to test and understand the efficiencies that could be found by using a particular data format for analysis.

In this paper five popular structured and semi-structured data formats (CSV, JSON, Parquet, ORC and Avro) are tested against a number of queries, using two different cloud-based technologies, to learn which format is the most efficient for certain forms of analysis.

This has resulted in understanding that ORC is the most efficient data format to use when using the Pig language over Apache Tez in Amazon EMR, amd while the CSV data format is more time efficient when used in Amazon Athena, ORC is the most inexpensive when considering monetary cost to run.

This paper is structured as follows; (1) an introduction to the background of the paper and the motivation behind the research, (2) an exploration of existing literature on the topic, (3) experiment and data preparation, (4) experiment implementation and (5) analysis and evaluation of experiment results.

Graham, H. and Peng, T.
A Comparison of the Efficiencies of Various Structured and Semi- Structured Data Formats in Data Analysis and Big Data Analytic Development.
DOI: 10.5220/0012756300003756

In *Proceedings of the 13th International Conference on Data Science, Technology and Applications (DATA 2024)*, pages 301-308
ISBN: 978-989-758-707-8; ISSN: 2184-285X

2 RELATED WORKS

Data is growing in volume at an ever-increasing rate. In 2013, it was reported that 2.5 quintillion bytes of data, or 2500 petabytes of data, were created each day (Wu et al., 2013). Ten years later, the Internet of Things, mobile devices, social media, sensors and a whole host of other technologies mean that the amount of data produced is growing rapidly, even when confined to a single field of expertise.

Volume of data is already a problem that many companies and organisations must consider, amongst others. While more data, particularly good quality, relevant data, is a good thing, it introduces additional challenges, such as processing and storage. Research and experimentation are constantly being instigated to address these challenges.

As previously stated, this paper will investigate structured (CSV) and semi-structured (JSON, Parquet, ORC and Avro) data formats. Apart from being structured or semi-structured, there is also a division between row-oriented (CSV, JSON and Avro) data structures and column-oriented (Parquet and ORC) data structures. This is also a consideration, in that the data a query is extracting, and where it is placed within, for example, a table, is important. If a sale or transaction is considered, in a row-oriented data format all the data about the sale, for example, the item, the cost, the date of sale, will all be stored together, and it will then store all the data for the next sale together. With a column-oriented data format, all the items sold will be stored together and the costs will be stored together. The data regarding each individual sale is still linked, but it is optimised to work with columns and subsets of columns, whereas row-oriented formats are optimised to work with and filter on individual rows, or entities (Dwivedi et al., 2012).

The compression of different data formats results in the size of the data on disk varying between the formats. Various data formats have been compared on this topic. JSON and CSV are not compressed whereas Parquet, ORC and Avro all make use of compression.

Related work shows that JSON is the largest of the five data formats on disk, followed by CSV (Belov et al., 2021).

Of the three compressed formats, Avro files usually take the most disk space (Belov et al., 2021a, 2021b; Naidu, 2022; Plase et al., 2017), however there are anomalies where Parquet data is the larger of the two (Abdullah & Ahmed, 2020). ORC consistently uses the least disk space (Belov et al., 2021a, 2021b; Pergolesi, 2019; Plase et al., 2017; Rodrigues et al., 2017). In one case, Parquet had a smaller footprint than ORC, but the difference was minimal (Naidu, 2022).

Related works, regarding which data formats are optimised for the functions being used in the later evaluation will be analysed. What follows is a brief summary of what these works discovered, based on data formats and technologies used.

Several studies showed that using a Spark-on-Hadoop environment, Parquet and ORC were the most efficient data formats (Belov et al., 2021a, 2021b), with Parquet out-performing ORC (Abdullah & Ahmed, 2020; Gupta et al., 2018; Pergolesi, 2019) and JSON being consistently the most inefficient data format (Belov et al., 2021a, 2021b).

Meanwhile, when using Hive as the underlying platform, ORC performed better than other data formats (Gupta et al., 2018; Naidu, 2022; Pergolesi, 2019). When ORC was not in consideration, it was shown that Parquet was a more efficient data format on Hive than Avro (Plase et al., 2016, 2017).

Avro generally proved to be an inefficient data format except when used with Impala (Gupta et al., 2018) and this data format and architecture combination was more performant than other combinations.

Finally, one study analysed Parquet and ORC using Amazon Athena and discovered that for lookup intensive queries, ORC was more efficient whereas for aggregating queries, Parquet was more efficient (Tran, 2019).

In comparison, this paper tests a range of individual functions using Amazon Athena and Amazon EMR across CSV, JSON, Parquet, ORC and Avro in addition to comparing overall performance using both time and monetary cost as evaluation metrics.

3 EXPERIMENT PREPARATION

The COVID Vaccination data from Our World in Data was chosen as the data on which to run experiments over. It was downloaded in JSON form, then modified to be usable.

Our World in Data, who have provided the data, make it their mission to provide “research and data to make progress against the world’s largest problems”. The COVID Vaccinations dataset (OWID, 2024) contains daily updates from each country regarding vaccinations, including total number of people vaccinated, daily vaccinations and booster vaccinations. There are fields for Country and ISO Code, and a nested data structure that can contain up

to twelve fields which may or may not be populated. This data structure is repeated for each day, with new values.

The data was originally published in order to allow data scientists to use it to monitor how effective vaccinations in a country, or globally, were, however it has been chosen for use in this experiment, not for the data it contains, but because it is a large, semi-structured, nested dataset that contains no personal information of any individuals, making it an ideal test dataset.

The COVID Vaccination dataset was originally in a nested JSON structure that was 62MB in size. The data required no cleaning prior to use; however, it did need to be transformed to be compatible with the platforms used. It was also duplicated to ensure that the data was large enough to produce meaningful results from the experiments.


```
{
  "country": "Afghanistan",
  "iso_code": "AFG",
  "data": [
    {
      "date": "2021-05-11",
      "total_vaccinations": 504502,
      "people_vaccinated": 448878,
      "people_fully_vacinated": 55624,
      "daily_vaccinations": 13921,
      "total_vaccinations_per_hundred": 1.23,
      "people_vaccinated_per_hundred": 1.09,
      "people_fully_vacinated_per_hundred": 0.14,
      "daily_vaccinations_per_million": 338,
      "daily_people_vaccinated": 10994,
      "daily_people_vaccinated_per_hundred": 0.027
    },
    {
      "date": "2023-02-26",
      "total_vaccinations": 14743912,
      "people_vaccinated": 13195588,
      "people_fully_vacinated": 12476994,
      "total_boosters": 536752,
      "daily_vaccinations": 162922,
      "total_vaccinations_per_hundred": 35.85,
      "people_vaccinated_per_hundred": 32.08,
      "people_fully_vacinated_per_hundred": 30.34,
      "total_boosters_per_hundred": 1.31,
      "daily_vaccinations_per_million": 3961,
      "daily_people_vaccinated": 116493,
      "daily_people_vaccinated_per_hundred": 0.283
    }
  ]
}
```


Figure 1: Sample raw JSON data.

It was discovered early in the experimentation phase JSON Lines (JSONL) is more appropriate than standard JSON for use in Athena. In addition, due to the levels of nesting, Athena was unable to load the data. This was due to the 'data' field, which was defined as an array of structures and contained a structure for each day the data was collected in the specified country. In many cases, the 'data' field was larger than 35MB, which is the size limit for a field in Athena. To mitigate this, the data was partially flattened and converted into JSONL, with the following schema, as can be seen in Figure 2 and with the definition as can be seen in Table 1.

```
{
        'country':string,
        'iso_code':string,
        'data': {
                'date':'string,
                'total_vaccinations':long,
                'people_vaccinated':long,
                'total_vaccinations_per_hundred':double,
                'people_vaccinated_per_hundred':double,
                'daily_vaccinations':long,
                'daily_vaccinations_per_million':double,
                'daily_people_vaccinated':long,
                'daily_people_vaccinated_per_hundred':double,
                'people_fully_vaccinated':long,
                'people_fully_vaccinated_per_hundred':double,
                'daily_vaccinations_raw':long,
                'total_boosters':long,
                'total_boosters_per_hundred':double
        }
}
```

Figure 2: JSONL schema.

Table 1: Attributes per country in JSONL data schema.

Attribute	Meaning
Date	Date data collected.
Total Vaccinations	Running total of vaccination doses administered.
People Vaccinated	Running total of people who have received at least one dose of vaccination
Total Vaccinations Per Hundred	Total Vaccination per hundred of the country's population.
People Vaccinated Per Hundred	People Vaccinated per hundred of the country's population.
Daily Vaccinations	Vaccinations performed on Date.
Daily Vaccinations Per Million	Daily Vaccinations per million of the country's population.
Daily People Vaccinated	Number of people receiving first dose of vaccination on Date.
Daily People Vaccinated Per Hundred	Daily People Vaccinated per hundred of the country's population.
People Fully Vaccinated	Running total of people who have received all prescribed doses of vaccination.
People Fully Vaccinated Per Hundred	People Fully Vaccinated per hundred of the country's population.
Daily Vaccinations Raw	Daily change in total number of vaccination doses administered.
Total Boosters	Running total of booster vaccinations administered.
Total Boosters Per Hundred	Total Boosters per hundred of the country's population.

This data was then duplicated one hundred times to create a sizable amount of data to test on. This created a total dataset size of 5.2GB. It was also

converted into Parquet, ORC and Avro, and flattened to create a copy of the data as a CSV.

During the experiment phase using EMR, it was discovered that the JsonLoader for Pig would only recognise JSONL records where the entire schema was complete. The semi-structured JSONL data was adjusted to be fully structured, by including all possible fields and giving them a default value, in order to load all the data. As the only optional fields were of type long or double, 0 and 0.0 were used as the default values.

Table 2: Data sizes.

Data Format	Size
JSON (Semi-Structure /Structured)	5.2GB / 8.4GB
CSV	1388.8MB
Parquet	366.5MB
ORC	332.9MB
Avro	1354.3MB

Finally, in order to test the JOIN function, a small dataset of Country Names and various codes (World Population Review, 2023) was downloaded to join to the test data. This data was in CSV format and stored in S3.

In addition to this the Amazon EMR Cluster was set up with the following specifications;

- EMR Release - emr-6.12.0
- Applications - Hadoop 3.3.3, Pig 0.17.0, Tez 0.10.2
- Primary and Core Node - m5.xlarge (1 Task Node and 1 Core Node)

The size of the cluster was set manually and auto-scaling was disabled.

4 IMPLEMENTATION

4.1 Implementation of Queries in Amazon Athena

The first stage of implementation was to create all the necessary tables in Athena. Six tables were created in total. Five of these were copies of the data from the Data Preparation Stage, in JSONL, CSV, Parquet, ORC and Avro formats. The sixth table was created using the Country and Country Code CSV data, which was used to JOIN to the data being assessed.

The second stage of the experiment was to design a series of queries that would test a range of functions. Initially eleven queries were designed to do this, plus a variant on four of the queries for the data formats that included a layer of nesting. Each query uses one or more of the SQL functions. The primary function that is being tested against each data format, is listed in Table 3 below.

Table 3: Functions used in SQL with Athena.

Name	Primary Function
SELECT	SELECT
SELECT UNNEST	SELECT
WHERE =	WHERE
WHERE = UNNEST	WHERE
WHERE >	WHERE
WHERE > UNNEST	WHERE
WHERE LIKE	WHERE
WHERE LIKE UNNEST	WHERE
GROUP BY COUNT	GROUP BY
GROUP BY SUM	GROUP BY
ORDER BY	ORDER BY
COUNT	COUNT
SUM	SUM
JOIN	JOIN
JOIN DISTINCT	JOIN

For these queries, any query name that contains the word ‘UNNEST’ is specific to the nested data formats, JSONL, Parquet, Avro and ORC. UNNEST in this instance refers to flattening the data.

Finally, each query was run using the Athena Query Interface. Each query was logged with how many rows it returned and how long it took to run against each table of a separate data format.

4.2 Implementation of Queries in Amazon EMR

Unlike Athena, tables are not created in EMR. Data is instead read directly from S3 for each query, requiring a schema to be provided for every query.

Table 4: Functions used in Pig with EMR.

Name	Primary Function
SELECT	LOAD
SELECT UNNEST	LOAD
WHERE =	FILTER BY
WHERE = UNNEST	FILTER BY
WHERE >	FILTER BY
WHERE > UNNEST	FILTER BY
WHERE LIKE	FILTER BY
WHERE LIKE UNNEST	FILTER BY
GROUP BY COUNT	GROUP BY
GROUP BY SUM	GROUP BY
ORDER BY	ORDER BY
COUNT	COUNT
SUM	SUM
JOIN	JOIN
JOIN DISTINCT	JOIN

The first stage of this experiment was to create comparative queries using Pig for EMR as had been developed in SQL for Athena.

A new Step in EMR was created for each query and every query was run separately to avoid contention for resource skewing the results.

When a Step, or Query, was run, results were output as CSV Part Files in S3 and the stdout logs provided more information about CPU Time and a more detailed Time to Run. The number of results, CPU Task Time and Time to Run are all recorded for evaluation.

5 EVALUATION

5.1 Evaluation of Data Formats Using Amazon Athena

Table 5: Runtime of queries using Athena in milliseconds.

Function	CSV	JSONL	Parquet	ORC	AVRO
SELECT *	65325	123616	210735	218493	186882
SELECT UNNEST	N/A	49780	81641	79615	69358
WHERE =	1152	2890	1995	1796	7084
WHERE = UNNEST	N/A	2312	1728	1392	9445
WHERE LIKE	1550	3757	2362	2456	10678
WHERE LIKE UNNEST	N/A	2.585	3.453	1.602	8106
WHERE >	20877	51938	62228	72468	45177
WHERE > UNNEST	N/A	21561	31467	31190	25371
GROUP BYCOUNT	1448	2528	1267	906	7999
GROUP BY SUM	2584	3346	5352	3858	8753
ORDER BY	15013	15207	13648	14180	21618
COUNT	1064	1219	718	832	2176
SUM	2456	3418	4827	5758	11438
JOIN	11735	13758	14981	13009	15027
JOIN DISTINCT	1662	3041	1806	2092	6528

As can be seen in Table 5, the types of functions being performed vary widely in performance against different data formats.

In order to determine where a function performed better over a data format when that function has been used multiple times, the mean of each performance metric was used to calculate the overall performance against the initial list of functions.

Table 6: Average (mean) performance of functions against data formats in Athena in milliseconds.

	CSV	JSONL	Parquet	ORC	AVRO
SELECT	65325	86698	146188	149054	128120
WHERE	7860	14174	17206	18484	17644
GROUP BY	2163	3097	3815	3507	9397
ORDER BY	15013	15207	13648	14180	21618
COUNT	1256	1874	993	869	5088
SUM	2520	3382	5090	4808	10096
JOIN	6699	8400	8394	7551	10778

It can be seen in Table 5 and Table 6 that loading in data (SELECT) and filtering it (WHERE) are most efficiently performed on the flat CSV data structure. It can also be seen that flattening the nested data structures allows the queries to perform more optimally.

There are three queries which use the GROUP BY function. For those that also use the SUM function, flat CSV data once again proves to be the most efficient, and CSV remains the most efficient overall. However, when counting records, whether using GROUP BY or not, Parquet and ORC, the columnar based data formats, prove to be superior. It can be surmised from the results that GROUP BY and COUNT performed on ORC data is more efficient than GROUP BY performed on Parquet.

ORDER BY is a slow operation to perform, however Parquet proved to be the most efficient data format to perform it on.

JOIN is most efficient when performed on flat CSV data, however, when the DISTINCT operator is used in addition to this, it is seen that Parquet's performance improves from fourth most efficient to second most efficient.

On average, CSV appears to be the most efficient data format, whereas Avro performs the worst in all functions except SELECT and the WHERE > queries. With the WHERE > queries, Parquet and ORC perform worse than Avro, which can be explained by them being columnar data formats, not optimised for filtering based on row values. JSON also performs badly on this query when the data is un-flattened.

However, regarding Athena, there is a separate consideration to make, which is the cost to run queries. The cost of running queries on Athena is based on the amount of data scanned, not the resources needed to run the query, or the time taken to run it. This means that businesses using Athena must make a choice between timeliness of queries

running versus cost, as CSV is, as shown in Table 2, is the second largest of the five data formats, whereas ORC, once again, is the smallest. Reference (Tran, 2019) shows that ORC is recommended as the most efficient data format to use. This supports the findings that ORC is the more efficient data format, based on monetary cost to run on Athena.

5.2 Evaluation of Data Formats Using Amazon EMR

Table 7: Runtime of queries using EMR in milliseconds.

Function	CSV	JSONL	Parquet	ORC	AVRO
SELECT *	84018	109684	68833	61137	74242
SELECT UNNEST	N/A	108219	85781	63030	79326
WHERE =	56974	84154	50379	36260	52587
WHERE = UNNEST	N/A	88395	49805	35001	51293
WHERE LIKE	59233	86811	58041	51491	53151
WHERE LIKE UNNEST	N/A	93269	48112	46478	55632
WHERE >	70027	94114	58207	53363	60343
WHERE > UNNEST	N/A	96470	59615	55452	71225
GROUP BY COUNT	69737	123346	62492	69765	71655
GROUP BY SUM	84418	161499	83305	67244	87577
ORDER BY	78256	195317	96765	858	97878
COUNT	69816	102325	67943	53133	69886
SUM	97936	176495	10673	128380	113648
JOIN	83531	135198	90587	82892	103551
JOIN DISTINCT	55482	101199	54252	51987	64289

Despite Avro being designed to be performant with Hadoop-backed systems, it again proved to be less performant in comparison to other data formats.

In all cases except three, ORC proved to be more efficient.

To confirm this, the Task CPU time for each query was also recorded. Task CPU Time is a more accurate measurement of performance than Total Run Time. This is because it measures the time the processors are working and eliminates variables such as time spent waiting for resources. Due to this, the true evaluation of performance will be based on Task CPU Time.

Table 8: CPU tasktime of queries using EMR in milliseconds.

Function	CSV	JSONL	Parquet	ORC	AVRO
CREATE TABLE	N/A	N/A	N/A	N/A	N/A
SELECT *	292460	410000	227020	198280	270220
SELECT UNNEST	N/A	424760	222250	207170	290670
WHERE =	206140	306640	133290	082870	176840
WHERE = UNNEST	N/A	315040	111690	063930	168750
WHERE LIKE	208510	323450	142660	136140	184940
WHERE LIKE UNNEST	N/A	321920	111110	099120	165720
WHERE >	253150	356940	180676	150410	215780
WHERE > UNNEST	N/A	380060	171790	168120	259950
GROUP BY COUNT	277120	490340	205590	148550	237920
GROUP BY SUM	362570	640150	264270	244990	356170
ORDER BY	285980	728300	320220	307140	393200
COUNT	271460	385920	191520	179510	217450
SUM	465030	837290	417390	430160	522740
JOIN	205270	408700	178560	162080	250420
JOIN DISTINCT	183920	395770	136330	146550	216740

Once again, the mean Task CPU Time for each main function is calculated, where it has been used in multiple queries.

Table 9: Average performance of functions against data formats in EMR in milliseconds.

	CSV	JSONL	Parquet	ORC	AVRO
SELECT	292460	417380	224635	202725	280445
WHERE	222600	334008	141869	116765	195330
GROUP BY	368240	655926	295750	274566	372276
ORDER BY	285980	728300	320220	307140	393200
COUNT	274290	438130	198555	164030	227685
SUM	413800	738720	340830	337575	439455
JOIN	194595	402235	157445	154315	233580

This confirmed that queries run against ORC data were largely more efficient. The only exceptions were ORDER BY, where the flattened CSV data was a more efficient input, (GROUP BY) SUM, which showed that when using multiple SUM functions,

Parquet was more efficient, and JOIN (DISTINCT), where Parquet was also the more efficient format, however, ORC was more efficient for the GROUP BY function and the SUM function on average. As the JOIN query showed that ORC was the more efficient data format, it can be surmised that running DISTINCT over Parquet data in Pig is more efficient.

Another observation was that unlike with Athena, unnesting data did not make a sizable efficiency improvement. In fact, in some cases, it decreased the efficiency of the query.

It is perhaps unsurprising that ORC was more performant than Parquet, despite the majority of existing literature suggesting otherwise. This is down to the platform architecture used in these experiments, and that of the platform used in the vast majority of the previous experiments.

Many of the previous experiments were performed on SparkSQL, which, as noted by (Gupta et al., 2018) performs better when used with Parquet Data. As also noted by (Gupta et al., 2018; Tran, 2019) that Hive performed better when used with ORC data, which, like Pig in EMR, is run by default on Apache Tez.

6 CONCLUSION AND FURTHER WORK

For this evaluation of data formats and their efficiencies for querying data and creating big data analytics, fifteen queries were developed and run over data structured in five data formats, CSV, JSON, Parquet, ORC and Avro, using different frameworks and languages for querying data, including Hadoop and SQL on Amazon Web Services (AWS) Athena and EMR. Metrics such as time taken, and Task CPU time were gathered for analysis.

The metrics gathered from these experiments were analysed and it was discovered that flat data was more efficiently processed on Athena, and that by flattening the data early in the query, it improved performance, but that for the most part, the CSV format was the most efficient, with ORC and Avro proving to be the least efficient. However, Pig on EMR proved to be optimised for ORC as it almost always proved to be the most efficient, except when using the ORDER BY function. In this case, Avro and JSON proved to be the least efficient.

An additional consideration with cloud computing platforms is the cost to run queries and analytics. This was also discussed, and it was determined that whilst using EMR, the most efficient data format was also the least expensive from a monetary aspect, with Athena, the data format that was the most reduced in volume was the least expensive from a monetary viewpoint. This was not CSV, but instead ORC, which had proved to not be as efficient to query, though its performance did improve when the data was flattened.

To conclude, it is recommended that when using the services described above, ORC is the most cost-effective data format to use, and, when analysing data using Pig on EMR, the most efficient.

This work can be expanded upon and it is intended to replicate these experiments to fully understand data efficiency on EMR, using Hive and Spark. This can be expanded across other AWS Services, but also on similar platforms from other cloud providers. This work would provide a comprehensive overview of which data formats are most efficient and cost-effective for use on a variety of cloud platforms.

REFERENCES

Abdullah, T., Ahmed, A. (2020). Extracting Insights: A Data Centre Architecture Approach in Million Genome Era. In *Hameurlain, A., Tjoa, A. M., Transactions on Large-Scale Data- and Knowledge-Centered Systems XLVI (pp. 1-31).* Springer Berlin Heidelberg. 10.1007/978-3-662-62386-2_1.

Belov, V., Tatarintsev, A. V., & Nikulchev, E. V. (2021a). Comparative Characteristics of Big Data Storage Formats. In *Journal of Physics Conference Series, 1727(1).* 10.1088/1742-6596/1727/1/012005.

Belov, V., Tatarintsev, A., & Nikulchev, E. (2021b). Choosing a Data Storage Format in the Apache Hadoop System Based on Experimental Evaluation Using Apache Spark. In *Symmetry, 13(2).* 10.3390/sym130 20195.

Dwivedi, A. K., Lamba, C. S., & Shukla, S. (2012). Performance Analysis of Column Oriented Database versus Row Oriented Database. In *International Journal of Computer Applications, 50(14), 31-34.* 10.5120/7841-1050.

Gupta, A., Saxena, M., & Gill, R. (2018). Performance Analysis of RDBMS and Hadoop Components with Their File Formats for the Development of Recommender Systems. In *2018 3rd International Conference for Convergence in Technology (I2CT), 1-6.* 10.1109/I2CT.2018.8529480

Naidu, V. (2022). Performance Enhancement using Appropriate File Formats in Big Data Hadoop Ecosystem. In *International Research Journal of Engineering and Technology, 9(1), 1247-1251.*

OWID-Our World in Data. (2024, March)). *COVID Vaccinations Data*. Retrieved from. https://github.com/owid/covid-19-data/tree/master/public/data

Pergolesi, M. (2019). The impact of columnar file formats on SQL-on-hadoop engine performance: A study on ORC and Parquet. In *Concurrency and Computation Practice and Experience, 32(5).* 10.1002/cpe.5523.

Plase, D., Niedrite, L., & Taranov, R. (2016). Accelerating data queries on Hadoop framework by using compact data formats. In *2016 IEEE 4th Workshop on Advances in Information, Electronic and Electrical Engineering (AIEEE), 1-7.* 10.1109/AIEEE.2016.7821807

Plase, D., Niedrite, L., & Taranovs, R. (2017). A Comparison of HDFS Compact Data Formats: Avro versus Parquet. In *Mokslas – Lietuvos Ateitis / Science – Future of Lithuania, 9(3), 267-276.* 10.3846/mla.2017.1033.

Rodrigues, R. A., Filho, L., Gonçalves, G. S., Mialaret, L., Marques da Cunha, A., & Dias, L. (2017). Integrating NoSQL, Relational Database, and the Hadoop Ecosystem in an Interdisciplinary Project involving Big Data and Credit Card Transactions. In *Information Technology - New Generations. Advances in Intelligent Systems and Computing, 558.* 10.1007/978-3-319-54978-1_57.

Tran, Q. D. (2019). Toward a serverless data pipeline. *Unpublished.*

World Population Review (2023). *Country Codes 2023*. Retrieved from. https://worldpopulationreview.com/country-rankings/country-codes

Wu, X., Zhu, X., Wu, G.-Q., & Ding, W. (2013). Data mining with big data. In *IEEE Transactions on Knowledge and Data Engineering, 26(1), 97-107.* 10.1109/TKDE.2013.109.