Atlantis Highlights in Computer Sciences, volume 4
Proceedings of the 3rd International Conference on Integrated Intelligent Computing Communication & Security (ICIIC 2021)

Aadhaar Data Analysis Comparison in MapReduce, Hive and Spark

Roopa R[1,*] Varsha Ryali[2] Tejasvi Shrivastava[3] Syed Mahmood Nabeel Anwar[4]

[1,2,3,4] *BMS College of Engineering*
**Corresponding author. Email: roopa.ise@bmsce.ac.in*

ABSTRACT
Aadhaar with a 12-digit unique identification number of every Indian provides demographic and biometric information and is mandatory for various purposes like benefit transfer directly, healthcare, etc. Approximately Aadhaar details need to store 1.3 Billion Indians which attributes to the concept of big data. In this paper, the proposed hybrid model analyses the Aadhaar dataset w.r.t different research interrogations such as count of applicants based on gender, state-wise approved and by age type applicants. In the existing systems, Aadhaar data analyses are done either manually or in primitive SQL platforms which may take days to complete. In this paper, the focus is on Aadhaar data analysis using different distributed computing frameworks like MapReduce, Hive, and Apache Spark on top of Hadoop that could be used for the purpose of better decision-making by all government firms and we provide the valid conclusion that Apache Spark framework is efficient in terms of performance.

Keywords: *Aadhaar, Big data, MapReduce, Hadoop, Hive, Apache Spark.*

1. INTRODUCTION

The analysis of Aadhaar data plays a significant impact not only to make efficient decisions but also supports the organizations in determining their positions relative to the competitors. The issues of demographic and biometrics verification have been resolved with the Aadhaar card projects which could identify duplicate and fake records. Unique Identification Authority of India (UIDAI) ensured that an outstanding number of citizens who are underprivileged are brought under the UID system and were able to discover non-existent beneficiaries in the welfare schemes of governments for the underprivileged [2,6].

The Aadhaar data is maintained in the CSV file contains the attributes such as Resident's mobile number, Residents providing email, Registrar, Agency, Enrollment State with District as well as Sub District, Zip Code, Age and Gender, Aadhaar Generated, Enrollment Rejected. The task is to analyze and find in each state the count of generated Identities, calculate the Identities generated based on Enrollment Agency, and identify the top ten districts that have the highest or max identities generated for both the Male and Female [10, 11, 14, 15].

In this paper, the Aadhaar data analysis is carried out on different distributed computing frameworks mainly MapReduce [1], Hive [3] and Apache Spark [4] on top of Hadoop.

A detailed comparison is provided on these and arrived at a conclusion which would be better for big data analysis like the Aadhaar dataset.

In this work, the approaches of distributed computing framework considered to perform Aadhaar data analysis are MapReduce, Hive, and Spark which are differentiated as shown in below tables:

Table 1. MapReduce, Hive and Apache Spark definitions

Mapreduce Framework	Hive	Apache Spark
Hadoop MapReduce is a software framework that processes multora byte data similarly in large-scale hardware clusters.	Hive is a data-based data processing tool that processes structured data at Hadoop Hive using SQL-inspired language, protecting the user from the complexities of MapReduce programming.	Apache Spark is a data processing framework that can quickly perform processing tasks on very large data sets, and can also deploy data processing functions on multiple computers either individually or in conjunction with other computer navigation tools. These two qualities are the key to the world of big data and machine learning that requires the marketing of large computer power to explode in big data stores. Spark also removes some of the programming loads of these tasks on the shoulders of developers with an easy-to-use AP that removes a lot of punt work for computer distribution and big data processing.

Table 2. MapReduce, Hive and Apache Spark approaches

Mapreduce Approach	Hive Approach	Apache Spark Approach
1.The MapReduce framework has a single master Job Tracker and a slave TaskTracker per cluster-node. 2.The master responsibilities are scheduling the jobs component tasks on the slaves, monitoring and re-executing the failed 3.Minimally the input/output locations are specified by applications and supply map and reduce functions through the Implementations of appropriate interfaces and/or abstract classes 4. The Hadoop job client submits the job (jar/executable etc.) and configuration to the JobTracker which then assumes the responsibility of distributing the software configuration to the slaves, scheduling tasks and monitoring them, providing status and diagnostic information to the job client.	1. Databases and tables are created first, and then the data is uploaded to the appropriate table 2. The driver works with the query compiler to find the program, which contains the query process and metadata details. The driver also analyzes the query to check syntax and requirements. 3. The compiler creates a metadata for the program to be executed and contacts the metastore to obtain a metadata application. 4. The driver sends the execution plans to the execution engine. 5. The Execution Engine (EE) processes the question by acting as a bridge between Hive and Hadoop. The work process works in MapReduce. The performance engine sends the function to JobTracker, which is located in the name node, and assigns it to Task Tracker, to the data node. While this is happening, the output engine performs metadata and metastore functions. 6. Results are obtained from data notes 7.The results are sent to the output engine, which, in turn, returns the results to the driver and to the front (UT).	1.Using spark-submit, the user submits the request 2. In spark-submission, please specify the main method the user specifies. It also introduces a driver program 3.The pilot program asks for resources from the group manager. We need to introduce the executors. 4. The collection manager opens up the bad guys instead of the driving system 5.The driver process works with the help of the user application. Depending on the actions and changes in the RDD, the driver sends the work to the providers in the form of tasks. 6. The performance and effect process is returned to the driver by the collection manager

Table 3. MapReduce, Hive and Apache Spark limitations

Mapreduce Limitations	Hive Limitations	Apache Spark Limitations
1.This approach requires lots of manual coding for simple operations like counting and sorting. 2.It doesn't support real-time processing and has no caching mechanism. 3.It would be suitable for few queries but processing a larger number of queries would require lots of lengthy code. 4.It is not easy to use when compared with other high-level programming frameworks like HIVE and Spark where the abstraction layer makes things simpler.	1.Hive requires simple Hive query language for simple operations and sorting 2.It supports real-time processing. 3.It does not support update and delete operation on tables. 4.Subqueries are also not supported. 5.No support for row-level updates or deletes.	1.No File Management System 2.No Real-Time Data Processing 3.Expensive 4.Small Files Issue 5.The lesser number of Algorithms Latency 6.Handling Back Pressure 7.Window Criteria 8.Manual Optimization

2. RELATED WORK

In [9] the authors have provided the data analysis comparison of Python and Scala programming languages based on parameters in particular with Apache Spark. With experimental evaluations and comparisons, the authors have concluded that selection of Python or Scala programming languages in Apache Spark depends on their project features.

The authors in [12] provided the analysis by extracting the output from HIVE and SPARK into excel and were visualized by plotting the data using line and bar plot charts. In this work HDFS is employed for storing huge amounts of airline data, Spark uses SparkSQL on the Spark framework and Hive uses Hive QL statements that run on MapReduce framework for querying the data.

In [5], the authors have shown that from the experimental analysis, Hadoop, which is an On-disk computation-based model, is lacking behind in terms of performance than the In-memory based computation model that's Spark. But when it comes to distributed environments, MapReduce cannot resolve the functionalities. But MapReduce is still used in fields of research for data manipulation and experimentation. Spark is up-to-date and has many additional features, especially In-memory data processing. Hence, Spark is used in fields of real-time processing.

In [13] the work carried out by the authors has introduced two case studies, one using GeoSpark and another employing Spatial Hadoop. They propose user-centric guidelines. These case studies describe applications whose requirements support real-world problems and whose datasets are extracted from the real-world sources OpenStreetMaps and Twitter, respectively. For every case study, they describe details regarding its data loading and indexing, show the execution of three unplanned spatial queries of interest, and supply the visualization of the results of those queries [16-17].

3. PROCEDURAL STEPS OF PROPOSED APPROACHES

In this work, a Hybrid model is proposed to perform Aadhaar data analysis which is carried out and compared with the following distributed computing framework:

i. MapReduce,
ii. Hive, and
iii. Apache Spark

3.1 MAPREDUCE APPROACH

The process of finding the number of Identities generated in Aadhaar w.r.t each state with the results sorted in descending order of count of identities using MapReduce framework are as follows:

Setup required

(i) Single node Hadoop cluster (if input data is small like in this particular case) otherwise multi-node Hadoop cluster for a huge amount of data
(ii) Eclipse IDE for writing MapReduce code
Three external jars are to be added to the java project are as follows:
a. commons-CLI
b. Hadoop-core
c. log4j
(iii) Input CSV file stored in HDFS

Algorithm

i. In all, there will be two MapReduce jobs. The 1st Map Reduce job calculates the total number of Aadhaars generated for each and every state. The 2nd

MapReduce job according to the number of Aadhaars generated sorts the States in descending order.

ii. Job 1 mapper takes the input CSV file where the input key is the byte offset of the line (except line 0 which is the header) and the input value is the contents of the line. Then the input value is split and the value corresponding to the "state" column is emitted as the output key whereas the value corresponding to "Aadhaar generated" is emitted as the output value.

iii. The intermediate outputs are then shuffled and sorted and transferred to the reducer by default. The reducer thus receives key as a state and value is of type list as all the values of the same keys are grouped together in respective partitions.

iv. Then the Reducer takes ("state", list of "Aadhaars generated") as key-value input pair. It then adds up the number of Aadhaars that are generated for every key–state and finally writes the aggregated pair of key-value to the output. So the output of the reducer of the first job is ("state", the total count of "Aadhaar generated") as output key-value pair.

v. Now second job's mapper takes job 1 reducer's output (state, count) as input key-value pair and just swaps key and value and emits output key-value pair as (count, state) to sort the count but by default MapReduce sorts according to keys, therefore, bring the count to key position temporarily to let the sort comparator sort it according to count.

vi. Now again shuffling and sorting of intermediate output takes place but since the default sorting comparator sorts in ascending order, we need to implement a custom sorting comparator that sorts in descending order so the compare method will be over-ridden accordingly.

vii. Finally, the reducer swaps back the input key-value pair (count, state) to the output key-value pair (state, count) and the final result is state-wise Aadhaar ids count in descending order [18-21].

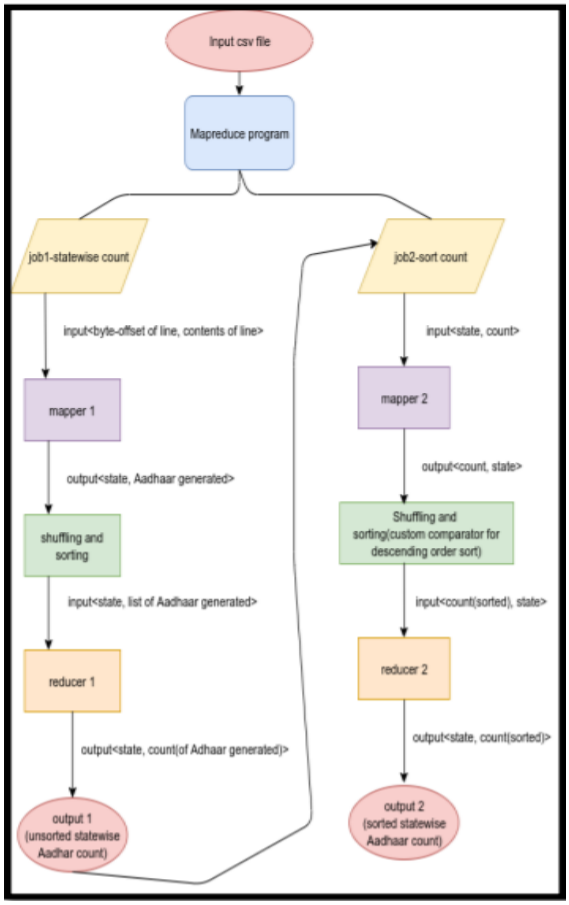

Figure 1 Data flow of the Algorithm

Steps of implementation

i. Open the Eclipse-> click on File -> then New -> Java Project ->(Name it – My Aadhaar) > Finish.

ii. Right Click -> click on New -> choose Package (Name it - demo) > Finish.

iii. Right Click on Package > New > Class (Name it - Aadhaar).

iv. Add the Following Reference Libraries:

v. Right Click on the Project- > choose Build Path-> Add External JARs

vi. Type the code mapper and reducer functions and the most important part of it is defining the driver code which serves as the entry point of the MapReduce job. It is where the input and output file locations will be set by making use of three run time parameters such as the input path, the output path for the First MapReduce job, and the output path for the second MR job. Apart from the normal driver configurations setting, it's required to create two more job objects. The first job object – stateWiseCount which implements the State-wise count mapper and classes of reducer whereas the second job object –implements the sorting

comparator, mapper, and reducer, classes that sort the output in non-ascending order.

vii. Create a jar file. Right Click on the Project-> then Export-> Select the export destination as Jar File -> choose next->choose Finish.

a) Start the cluster.
b) Take the input CSV file and move it into HDFS format.
c) Run the jar file:
Open output directory of job 1>open part-r-00000 file Here the no. of Aadhaar ids are not sorted in descending order.
Open output directory of job 2>open part-r-00000 file:
Finally, the output is the count of Aadhaar ids per state sorted in descending order of the count of Aadhaar ids.

3.2. HIVE APPROACH

The procedural steps of Aadhaar data analysis using the HIVE framework are as follows:

Derby is an open-source relational database management system. It provides users fine-grained access rights according to SQL standards and with standards of small footprint that are based on a database engine that can be embedded tightly into any of the Java-based solutions.

After successfully installing Hive and setting up Derby on Ubuntu. [22-25]

Steps of Implementation

Starting with the adhaar.csv file contains the following columns: Registrar details, Registration Agency, Region, Sub-District, Zip Code, Gender, Age, Aadhaar Generated Details like, Date of Registration, Residents Providing email, mobile number.

And when the data is ready, look at the nest query to get a calculation of the different identities generated in each state. As the hive works internally with the MapReduce, the MapReduce snippet is walked through for it is required to calculate the identities generated state-wise.

Creation of a hive table with extracting adhaar.csv

Figure 2 Create a Table query.

The above query as shown in Figure 2 creates a table Named Aadhaar with the data defined.

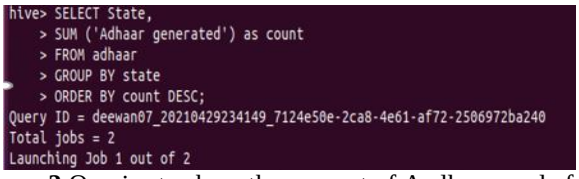

Figure 3 Queries to show the amount of Aadhaar made for all state.

Initial Mapreducer work includes counting Aadhaars produced by each state. Second Map Reduce program performs the activity by sorting states according to the number of Aadhaars generated by the them in decreasing order

Output

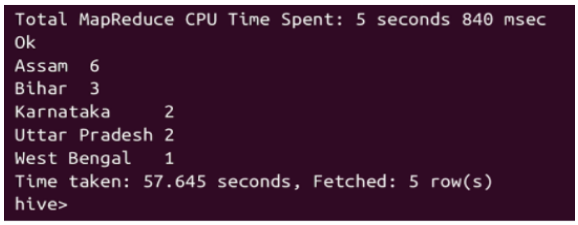

Figure 4 The output with the time taken to generate the data.

The above results are the total of Aadhaar generated in every state, in decreasing order.

A number of Aadhaar Identities which are generated by each of the Enrollment Agency is as shown below:

```
hive> SELECT 'enrolment agency' as Enrolment_Agency,
    > count ('Adhaar generated') as count
    > FROM adhaar
    > GROUP BY Enrolment_Agency
    > ORDER BY count DESC;
Query ID = deewan07_20210430000619_a9856a70-82f0-4cb1-802b-23c9027c3404
Total jobs = 2
Launching Job 1 out of 2
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapred.reduce.tasks=<number>
```

Figure 5 Query to find the total count of Aadhaar Identities which are generated by each of the agencies of Enrollment.

Output

```
Total MapReduce CPU Time Spent: 4 seconds 910 msec
OK
enrolment agency        18
enrolment agency        16
enrolment agency        12
enrolment agency        6
enrolment agency        1
enrolment agency        1
enrolment agency        1
enrolment agency        1
enrolment agency        1
Time taken: 40.346 seconds, Fetched: 9 row(s)
hive>
```

Figure 6 The output results of the total count of Aadhaar generated by each Enrollment Agency, in decreasing order w.r.t time taken.

```
hive> SELECT District,
    > count(CASE WHEN Gender='M' THEN 1 END) as male_count,
    > count(CASE WHEN Gender='F' THEN 1 END) as female_count
    > FROM adhaar
    > GROUP BY District
    > ORDER BY male_count DESC,female_count DESC
    > LIMIT 10;
Query ID = deewan07_20210430001059_537652b6-d21d-4b32-90ff-10d4cda348b9
Total jobs = 2
Launching Job 1 out of 2
```

Figure 7 Query to find the Top Ten districts with a max of Aadhaar identities produced for both genders.

```
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 2.79 sec   HDFS Read: 15657 HDFS Write: 442 SUCCESS
Stage-Stage-2: Map: 1  Reduce: 1   Cumulative CPU: 2.43 sec   HDFS Read: 5798 HDFS Write: 140 SUCCESS
Total MapReduce CPU Time Spent: 5 seconds 220 msec
OK
Lakhisarai      18      0
Bagalkot        7       3
Varanasi        5       0
Bara Banki      1       0
Gopalganj       1       0
Gorakhpur       1       0
Marigaon        1       0
Shamli  1       0
Sonbhadra       1       0
Cooch Behar     0       16
Time taken: 43.672 seconds, Fetched: 10 row(s)
hive>
```

Figure 8 The results w.r.t time taken to generate top ten districts with a max of Aadhaar identities for both the Male as well as Female, in the decreasing order.

3.3 Spark Approach

The installation is based on the Linux Operating System. It consists of the Java installation with the environment variables along with Apache Spark. The recommended prerequisite installation is Python, which is done from here. (latest python version i.e. python 3.9 is used)

Setup Required

1. Jupyter Notebook: To have a separate environment for pyspark, Docker is used to making a separate container for pyspark

```
[ ] import findspark
    findspark.init()

    findspark.find()

    '/content/spark-3.1.1-bin-hadoop2.7'

[ ] from pyspark.sql import SparkSession
    from pyspark.sql import Row
    import collections

[ ] spark = SparkSession.builder\
            .master("local")\
            .appName("Colab")\
            .config('spark.ui.port', '4050')\
            .getOrCreate()

[ ] spark
```

SparkSession - in-memory

SparkContext

Spark UI

Version
v3.1.1
Master
local

Figure 9 SparkSession Initiation

Spark Session is an integrated entry point from Spark 2.0. spark application. It offers a bridge of interaction with various sparks with a small amount of construction. Instead of spark context Hive context, SQL context, now it's all included in the Spark session.

Create a new session builder if not previously created and grant the newly created Spark Session as a global default.

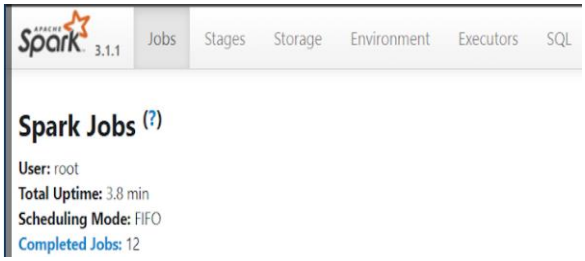

Figure 10 (a)Job details.

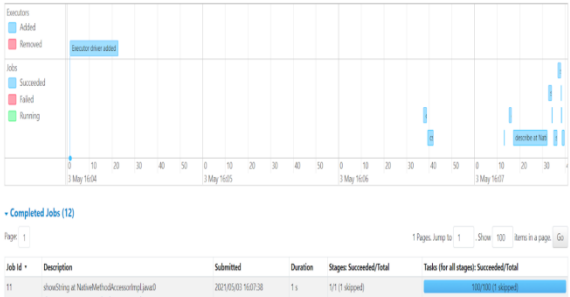

Figure 10 (b) Time taken in Spark Session for the Aadhaar related jobs submitted.

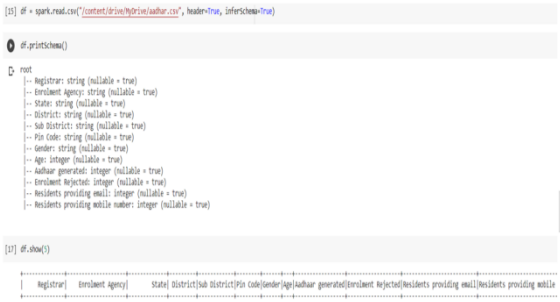

Figure 11 The figure represents the Loading and understanding dataset

The Following query was performed for the dataset shown in Figure 10:

1. Sort States according to the total count of Aadhaar generated.

```
df.groupBy("state").agg(F.sum("Aadhaar generated")).show()
```

```
+--------------------+-----------------------+
|               state|sum(Aadhaar generated)|
+--------------------+-----------------------+
|            Nagaland|                    545|
|           Karnataka|                  19764|
|              Odisha|                  18182|
|              Kerala|                  15143|
|          Tamil Nadu|                  32485|
|        Chhattisgarh|                   6604|
|      Andhra Pradesh|                   5798|
|         Lakshadweep|                      4|
|      Madhya Pradesh|                  53276|
|              Punjab|                   6506|
|             Manipur|                   1323|
|                 Goa|                   1167|
|             Mizoram|                   6279|
|Dadra and Nagar H...|                    140|
|    Himachal Pradesh|                   1547|
|          Puducherry|                     83|
|             Haryana|                   6804|
|   Jammu and Kashmir|                   1234|
|           Jharkhand|                   9868|
|   Arunachal Pradesh|                    913|
+--------------------+-----------------------+
only showing top 20 rows
```

Figure 12 Result of the sort query is shown.

2. Queries for obtaining the Aadhaar ID number generated by each Registration Agency and its outcome as shown below:

```
df.groupBy("Enrolment Agency").agg(F.sum("Aadhaar generated")).show()
```

```
+--------------------+-----------------------+
|    Enrolment Agency|sum(Aadhaar generated)|
+--------------------+-----------------------+
|Raj Construction Co.|                    532|
|      CO JOMLO MOBUK|                      8|
|NPS Technologies ...|                   9692|
|    APOnline Limited|                    305|
|         DC Serchhip|                   1067|
|District Magistra...|                     50|
|Quick Data IT Ser...|                      1|
|Directorate of Wo...|                    209|
|              WEBEL|                      0|
|  Directorate of ESD|                   5514|
|Omnitech Infosolu...|                   2032|
|  Transmoovers India|                      5|
|Zephyr System Pvt...|                   6946|
|          ADC BOLENG|                      2|
|District IT Socie...|                     29|
|Emdee Digitronics...|                   2078|
|Offshoot Agency P...|                     91|
|Netlink software ...|                   4832|
|     DSO STAT NAMSAI|                     50|
|District Sukhmani...|                      5|
+--------------------+-----------------------+
only showing top 20 rows
```

Figure 13 Result of the query for count is shown.

3. Query to find the Districts with highest/max Aadhaar identities generated for both the Male as well as females and the output as shown:

```
df.crosstab("District","Gender").show()
```

```
+---------------------+---+----+---+
|     District_Gender|  F|   M|  T|
+---------------------+---+----+---+
|            Chandauli|164| 418|  0|
|South West Khasi ...|  0|   1|  0|
|         Baloda Bazar| 79| 181|  0|
|               Panna|159| 327|  0|
|      East Champaran|966|2099|  0|
|           Puruliya|721| 762|  0|
|              Siang|  6|  32|  0|
|        Kamrup Metro| 44| 150|  0|
|               Surat|692|1270|  0|
|               Unnao|260| 529|  0|
|              Mandla|105| 259|  0|
|               Jalna|115| 179|  0|
|       Pauri Garhwal| 98| 360|  0|
|              Khunti| 25|  46|  0|
|             Nellore| 42| 143|  0|
|              Kanker| 38|  59|  0|
|          Villupuram|228| 457|  0|
|            Azamgarh|450|1017|  0|
|             Kurnool|107| 240|  0|
|             Koderma| 59|  89|  0|
+---------------------+---+----+---+
only showing top 20 rows
```

Figure 14 Result of the query for sort district-wise in descending order is shown [26-29].

4. RESULTS

A comparative analysis of the proposed work and existing works of [7] and [8] leads concludes that Spark is the favorable option of the three.

In [7], authors uploaded word count and Transport functionality at 18 parameter values by replacing the default set. To investigate performance, they used the trial and error method to fix these components that make up the test number

in a nine-node cluster with a capacity of 600 GB databases. There is Hadoop due to its ability to process Spark data in memory instead of map disk store and reduced performance. They found that Spark performance was reduced when input data was large.

In [8], It uses Hive to manage common queries such as SQL extending its use to structured data and the effects of low latency. Nest can generally be a preferred method in processing large-sized data sets, especially when costs are high, energy efficiency, and rawness are important. It is usually better when data volumes are replaced by regular real-time processing. The nest may also be preferred when data is distributed in multiple locations

Combining the learnings of this existing analysis, a detailed conclusion of this paper's venture on Map-reduce, Hive and Apache Spark are:

A. MapReduce

The map-reduce approach required lots of manual coding for simple operations like counting and sorting. Also, it doesn't support real-time processing because of latency due to its inherent architecture and absence of caching mechanism. It would be suitable for a few queries but processing a larger number of queries would require lots of lengthy code. Additionally, it is not easy to use when compared with other high-level programming frameworks like HIVE and Spark where the abstraction layer makes things simpler.

B. Hive

Hive is not able to process and handle a large set of Aadhaar data. Proved out to be accurate and reliable in handling compact datasets. It is quite faster in running queries. Takes very little time to write a Hive query in comparison to MapReduce code.

C. Apache Spark

Query results and file operations were quite fast as compared to that of hive and map-reduce but Apache Spark does not have its own file management system, so it relies on another platform like Hadoop or another cloud-based platform that is one of Spark's known issues. One of the strongest factors in support of Apache spark is it supports Scala, Java, Python, SQL, and R.

Among the three approaches Apache spark came out to be the most suitable for handling large data sets like Aadhaar, Apache spark is able to handle large sets of data and process it in one go.

Table 4. Comparative Analysis of the results:

Parameters	MapReduce	Hive	Spark
Approach	Java programs	HQL	Pyspark
Time taken for analysis	~45 sec	~3-5 sec	~0.1- 1.0 sec

5. CONCLUSION

The Paper presents different distributed computing approaches to extract meaningful knowledge and analyze the Aadhaar data from the Aadhaar dataset. The 3 main frameworks used for extracting data were MapReduce, Hive, and Apache Spark performed on single node cluster Hadoop. All of the three have their own procedural steps and limitations too. In comparison with the three approaches Apache spark came out to be the most suitable for handling large data sets like Aadhaar, Apache spark is able to handle large sets of data and process it in one go. As MapReduce and hive are performing on single node cluster Hadoop, they are found to be quite inefficient to handle large datasets. For future work, Multi-node clusters can be incorporated and also Super computers with high-end specifications through which a large number of datasets can easily manage and handle and process the results more efficiently and accurately for applications with huge amounts of data processing which is useful for better decision making by the state and central government.

REFERENCES

[1] N. Ahmed, A.L.C. Barczak, T. Susnjak, M.A. Rashid, "A comprehensive performance analysis of Apache Hadoop and Apache Spark for large scale datasets using HiBench", vol. 7, no. 1, pp. 110, 2020. https://doi.org/10.1186/s40537-020-00388-5

[2] X. Ji, M. Zhao, M. Zhai, Q. Wu, "Query Execution Optimization in Spark SQL", Scientific Programming, vol. 2020, pp. 6364752, Feb 2020. https://doi.org/10.1155/2020/6364752

[3] M Banane, A Belangour, "A new system for massive RDF data management using Big Data query languages Pig, Hive, and Spark", International Journal of Computing and Digital Systems, 9(2), 259-270, 2020. DOI: http://dx.doi.org/10.12785/ijcds/090211, Date: 2020-03-01

[4] Z.H. Jin, H. Shi, Y.X. Hu, L. Zha, X. Lu, "CirroData: "Yet Another SQL-on-Hadoop Data Analytics Engine

with High Performance", Journal of Computer Science and Technology volume 35, no. 1, pp. 194–208 (2020)

[5] S. Ketu, P.K. Mishra, S. Agarwal, "Performance Analysis of Distributed Computing Frameworks for Big Data Analytics: Hadoop Vs Spark", Computación y Sistemas, 24(2), (2020).

[6] M. Ragab, R. Tommasini, S. Eyvazov, S. Sakr, "Towards making sense of Spark-SQL performance for processing vast distributed RDF datasets", In Proceedings of The International Workshop on Semantic Big Data, pp. 1-6. 2020. -Published:14 June 2020

[7] K. Anusha, K.U. Rani, "Performance Evaluation of Spark SQL for Batch Processing", In Emerging Research in Data Engineering Systems and Computer Communications, pp. 145-153. Springer, Singapore, 2020. Published: 11 February 2020

[8] H.E. Ciritoglu, J. Murphy, C. Thorpe, "Importance of Data Distribution on Hive-Based Systems for Query Performance: An Experimental Study", In 2020 IEEE International Conference on Big Data and Smart Computing (BigComp) (pp. 370-376). IEEE. Published: 20 April 2020

[9] "A Study of Big Data Analytics using Apache Spark with Python and Scala" Published:18 January 2021

[10] Y.K. Gupta, S. Kumari, "Performance Analysis of Sales Big Data Processing using Hadoop and Hive in Cloud Environment", In 2020 3rd International Conference on Intelligent Sustainable Systems (ICISS) (pp. 471-478). IEEE. Published: 21 January 2021

[11] A.C. Phan, T.C. Phan, T.N. Trieu, "A Comparative Study of Join Algorithms in Spark", In International Conference on Future Data and Security Engineering (pp. 185-198). Springer, Cham. Published: 19 November 2020

[12] S. Chourawar, "Performance Comparison Between Hive-QL and Spark-SQL on Analysis of Airlines Dataset" - International Journal of Advanced Research in Computer and Communication Engineering Vol. 9, Issue 5, May 2020.

[13] JP de Carvalho Castro, A Chaves Carniel, C. Dutra de Aguiar Ciferri, "Analyzing spatial analytics systems based on Hadoop and Spark: A user perspective," Software: Practice and Experience, 50(12), 2020 December, pp. 2121-2144.

[14] A. Adesokan, "Performance Analysis of Hadoop MapReduce And Apache Spark for Big Data", https://osuva.uwasa.fi/handle/10024/11369 Published:2020-09-10

[15] P.N. Saipraveen, G.S. Nagaraja, "Parameter Tuning of Apache Spark-based Applications for Performance Enhancement", International Research Journal of Engineering and Technology (IRJET), vol. 7, no. 5, pp. 3491-3495 - Published:05 |May 2020

[16] K. Yu, L. Tan, L. Lin, X. Cheng, Z. Yi and T. Sato, "Deep-Learning-Empowered Breast Cancer Auxiliary Diagnosis for 5GB Remote E-Health," IEEE Wireless Communications, vol. 28, no. 3, pp. 54-61, June 2021, doi: 10.1109/MWC.001.2000374.

[17] K. Yu, L. Tan, S. Mumtaz, S. Al-Rubaye, A. Al-Dulaimi, A. K. Bashir, F. A. Khan, "Securing Critical Infrastructures: Deep Learning-based Threat Detection in the IIoT", IEEE Communications Magazine, 2021.

[18] K. Yu, Z. Guo, Y. Shen, W. Wang, J. C. Lin, T. Sato, "Secure Artificial Intelligence of Things for Implicit Group Recommendations", IEEE Internet of Things Journal, 2021, doi: 10.1109/JIOT.2021.3079574.

[19] H. Li, K. Yu, B. Liu, C. Feng, Z. Qin and G. Srivastava, "An Efficient Ciphertext-Policy Weighted Attribute-Based Encryption for the Internet of Health Things," IEEE Journal of Biomedical and Health Informatics, 2021, doi: 10.1109/JBHI.2021.3075995.

[20] L. Zhen, Y. Zhang, K. Yu, N. Kumar, A. Barnawi and Y. Xie, "Early Collision Detection for Massive Random Access in Satellite-Based Internet of Things," IEEE Transactions on Vehicular Technology, vol. 70, no. 5, pp. 5184-5189, May 2021, doi: 10.1109/TVT.2021.3076015

[21] L. Tan, K. Yu, A. K. Bashir, X. Cheng, F. Ming, L. Zhao, X. Zhou, "Towards Real-time and Efficient Cardiovascular Monitoring for COVID-19 Patients by 5G-Enabled Wearable Medical Devices: A Deep Learning Approach", Neural Computing and Applications, 2021, https://doi.org/10.1007/s00521-021-06219-9

[22] Puttamadappa, C., and B. D. Parameshachari. "Demand side management of small scale loads in a smart grid using glow-worm swarm optimization technique." Microprocessors and Microsystems 71 (2019): 102886.

[23] Parameshachari, B. D., H. T. Panduranga, and Silvia liberata Ullo. "Analysis and computation of encryption

technique to enhance security of medical images." In IOP Conference Series: Materials Science and Engineering, vol. 925, no. 1, p. 012028. IOP Publishing, 2020.

[24] Rajendran, Ganesh B., Uma M. Kumarasamy, Chiara Zarro, Parameshachari B. Divakarachari, and Silvia L. Ullo. "Land-use and land-cover classification using a human group-based particle swarm optimization algorithm with an LSTM Classifier on hybrid pre-processing remote-sensing images." Remote Sensing 12, no. 24 (2020): 4135.

[25] Hu, Liwen, Ngoc-Tu Nguyen, Wenjin Tao, Ming C. Leu, Xiaoqing Frank Liu, Md Rakib Shahriar, and SM Nahian Al Sunny. "Modeling of cloud-based digital twins for smart manufacturing with MT connect." Procedia manufacturing 26 (2018): 1193-1203.

[26] Seyhan, Kübra, Tu N. Nguyen, Sedat Akleylek, Korhan Cengiz, and SK Hafızul Islam. "Bi-GISIS KE: Modified key exchange protocol with reusable keys for IoT security." Journal of Information Security and Applications 58 (2021): 102788.

[27] Nguyen, Tu N., Bing-Hong Liu, Nam P. Nguyen, and Jung-Te Chou. "Cyber security of smart grid: attacks and defenses." In ICC 2020-2020 IEEE International Conference on Communications (ICC), pp. 1-6. IEEE, 2020.

[28] Arun, M., E. Baraneetharan, A. Kanchana, and S. Prabu. "Detection and monitoring of the asymptotic COVID-19 patients using IoT devices and sensors." International Journal of Pervasive Computing and Communications (2020).

[29] Kumar, M. Keerthi, B. D. Parameshachari, S. Prabu, and Silvia liberata Ullo. "Comparative Analysis to Identify Efficient Technique for Interfacing BCI System." In IOP Conference Series: Materials Science and Engineering, vol. 925, no. 1, p. 012062. IOP Publishing, 2020.