Volume 10, Issue 1
February 2021

ITEE Journal
Information Technology & Electrical Engineering

ISSN: - 2306-708X

Multi-Stage Change Data Capture for Files Based System Using Spark ETL Engine

[1]Mohammed Muddasir N, [2]Raghuveer K and [3]R Dayanand
[1]Asst. Prof, Dept. of IS&E, VVCE, Mysuru, India
[2]Principal, NIE, Mysuru, India
[3] Technical Director
E-mail: mohammed.muddasir@vvce.ac.in, raghunie@yahoo.com, Dayanand_7@yahoo.com

ABSTRACT

Real time analytics in the age of web2.0 comes with challenges particularly for data integration systems like Extract Transform Load (ETL) systems. ETL systems should adapt solutions based on distributed system and parallel processing to handle huge volumes of data having various formats and generated at a faster rate (big data). Spark based ETL engine has been used as an alternative for traditional ETL too to handle big data. Spark is used for data integration in a distributed environment where the data resides on cluster of computers like hadoop distributed files systems (HDFS). This paper focuses on change data capture (CDC) in a files based system without the support of CDC solutions provided by relational database management systems (RDBMS). In an enterprise RDBMS CDC solutions based on triggers, snap shots are provided and developers have to leverage on these to identify and move changed data. In a big data environment custom solutions are needed because the technological advancements to address specific issues like slowing changing dimension (SCD), CDC are limited. This works comes up with a CDC solution on spark ETL engine using the concept of self-join implemented on a scala shell for a file based system. We also propose and show results of a multi-stage CDC to reduce the efficiency in execution of self-join operations by reducing the number of comparison required to identify the modified data.

Keywords: *Change data capture, Data warehouse, big data, Extract Transform Load*

1. INTRODUCTION

Extract Transform Load (ETL) in the advent of big data brings in new challenges and requires novel solutions to address characteristics of big data such as volume, velocity and variety[1]. Extraction of data in transactional databases was driven by capabilities of the database management systems that would be tailored for data warehousing needs. Transformation was also dependent on custom codes that are built using structured query language (SQL) or using standardized data integration tools like Talend, Pentaho Kettle etc. Loading was with the support of data warehouse solutions provided by database management system vendors. Conventional ETL tools that were good for handling relational database do not suffice for handling big data and hence requires new solutions based on distributed file systems and Map-Reduce programming paradigm[2][3]. Focusing on data warehouse specific functionalities like slowly changing dimensions (SCD) and change data capture(CDC) requires special attention in case of big data.

This work is about CDC in big data using multi-stage approach. In case of structured sources transactional data management systems employ a push approach where in the changed data are extracted from the source using the tools and custom code for the particular data management system. Here the data extracted is given as a output file and it's the data warehouse job to integrate this file with the existing data. Change data capture for unstructured sources (big data) is based on the pull approach that is to query the source at regular intervals. Various ways to pull data to identify changed data [4] can be listed as follows; audit columns, row difference comparisons, modified time column for each of the sources etc. Every approach has a several challenges depending upon the information available. Example in times stamp based approach an atomic unit of data needs to be presented with a modified time. In hash based approach appropriate hashing functions needs to be available for pulling out the changed data. In this work we focus on time stamp based change data capture where in the assumption is every row that is modified shall be updated with the latest modified date and time. The problem we identified is in the execution efficiency of time stamp based CDC. The data from the source needs to be compared with the data warehouse contents to identify and move changed contents. This could be achieved using a staging area where newly arrived data is compared using self-join concept with the existing data in the stage and pick the rows that were newly added as well as those that were recently updated and move them all to the data warehouse. Here the issue is every time the new files arrives self-join needs to be executed.

The approach we propose is to reduce the number of times self-join is executed by waiting. Here we have to wait for two or more files to arrive and then combine their information into the staging area one and perform the self-join operation with the previous data in staging area two hence making it multi-stage CDC. This helps in improving the execution time and makes the process of performing

Volume 10, Issue 1
February 2021

ITEE Journal
Information Technology & Electrical Engineering

ISSN: - 2306-708X

CDC much faster. The results show that combined data files results in much better CDC timing than separate data files. We have implemented a spark based ETL engine where in data frames are used for holding the staged data. Combine two or more files into a single data frames before performing CDC. The idea of performing change data capture using self-join requires a comparison of n rows with another n rows of the same relation. If the data is present in a single data frame the self-join is done only once and it requires for n^2 comparisons. If combined approach is not used then the comparison has to be done for every new file. Suppose two files arrive separately then each file content is self-joined with staging area content twice, assuming equal rows in the two files arrived the comparison would be 2 times n^2. If the number of files increase so does the number of times comparisons that have to be done increases. Hence we show that multi-stage approach to wait for many files to arrive and then perform self-join to extract changed data is a better approach.

2. RELATED WORK

ETL and big data have been the focus of research for many academicians in recent times. Every work focuses on certain issues that could address challenges of faster execution, modeling, scalability etc.

A work using map reduce for handling ETL was done by authors of [5]. They use parallel dimensional frame work to address slowly changing dimensions (SCD). They combine offline and online dimension schema for better scalability. Offline dimension help to reduce direct interaction with the data warehouse they are stored on a distributed system and are partially brought into the main memory for a faster lookup during fact processing. Similar work by the same authors to address slowing changing dimension was called cloud based ETL[6].The novelty of cloud based ETL is co-locate data for a specific purpose. The authors of [7] have adopted hash based CDC where in two tuples are compared for equality of their keys and hash valued. If both the equations evaluate to true then the tuple in comparison would be rejected as it was not modified or added after the previous incremental load. However if the keys match and hash value do not match then it means the tuples have a different values in source and data warehouse and hence concluded to be modified. This modified tuples is the candidate for change data capture. The novelty of their approach is computing the hash value using cyclic redundancy check.

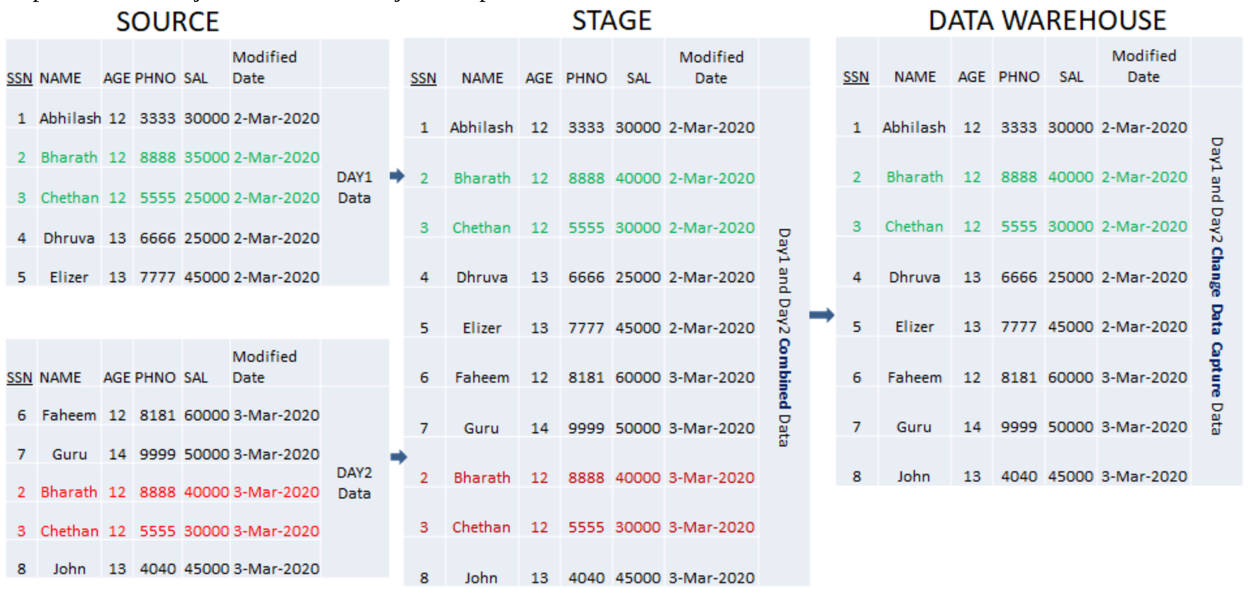
Partitioning and parallelization of ETL was done in [8] where a frame work consisting of data flow designer, practitioner, task planner component manager and meta data model are used to optimize the process. Modeling ETL to handle volume characteristic of big data was done by authors of [9]. They use anticipation during early stages of ETL for parallelization and distribution to increase processing speed. Transformation part of ETL is done using two phases map and reduce. The work by authors of[10] is about automatic scalability of ETL to handle small and big data. They have come up with architecture for automatic scalability which comprises of performance monitoring sub system, configuration sub system, universal data warehouse manager, scheduler and pool of nodes that could be added for scaling up and removed for scaling down. Handling data that arrive late to populate slowing changing dimension was proposed in [11]. They use two levels of staging because to handle late arriving or early arriving data needs a platform to rest. By having two staging areas ETL flexibility is achieved along with minimal intervention of the data warehouse.

The authors of [12] propose two phase map reduce to solve data warehouse problem related to big data. Joining dimension and facts are required to populate the data warehouse and hence if the data sizes are huge this is a performance bottle neck. Two phase map reduce to perform aggregations on mappers and processes intermediate results on reduces. Mappers and reduces are separated for facts known as fact mappers reducers, dimension mappers and reduces. CDC based on differential snap shot is implemented in [13]. Change data capture in big data based on snap shot requires to compare huge number of attributes and hence a time consuming process. The problem of increase in data volumes was handled using a distributed file system approach that was used to store full extracted data as silos on a hadoop distributed file system (HDFS). Outer join operations of SQL spark was used to implement CDC. Streamed data sources are handled using balance optimization techniques in [14]. Balanced optimizer minimizes data movement, uses optimization solutions available are source and destination, and maximizes parallel processing.

Through the literature survey we have explored on various aspects of ETL that are handled for big data by various authors through last decade. We could only find two papers related to CDC [7] [13], one paper using hash based function to identify changed row and other uses snap shot for identifying changed data. In this paper we use time stamp based CDC and also implement a novel two stage approach to reduce the comparison efficiency.

3. CHANGE DATA CAPTURE ILLUSTRATED

The techniques of change data capture (CDC) retrieves the data from the source that which is updated (insert, update, delete). In the below Fig1 shown is a glimpse of the scenario. Source has got two files Day1 data and Day2 data. Day1 data has 5 tuples and Day2 also has 5 tuples. Assuming that Day1 data is already in the data warehouse Day2 data needs to be combined with Day1 data. Tuples highlighted with red color in Day2 are those tuples that match with the existing tuples in Day1 that are highlighted in green color. The CDC would retain the latest entries for similar rows i.e. in this case tuples with SSN 2 and 3. Day1 data needs to be replaced for the tuples with SSN 2 and 3. Also the newly added data in Day2 needs to be present at the data warehouse. The implementation is achieved by combining the data from Day1 and Day2 into a staging area as a single table. Now staging area contains total of 10 tuples 5 each from Day1 and Day2. In the staging area the table is made to perform a self-join on SSN. Self-join output will be 10 tuples that match 10 SSN values each from two copies with 12 columns viz(SSN, Name, Age, PHNO, Sal, Modified data) from two copies of the table. Next is to group the data for second copy on SSN and select maximum value of date. Finally the first copy of the table is made to filter SSN that match the max modified date from the second copy of the table. Final result as seen in the data warehouse table is 8 rows, by eliminating repeated rows and retaining all the newly added rows.


SOURCE
DAY1 Data
DAY2 Data
STAGE
Day1 and Day2 Combined Data
DATA WAREHOUSE
Day1 and Day2 Change Data Capture Data

SOURCE — DAY1 Data

SSN	NAME	AGE	PHNO	SAL	Modified Date
1	Abhilash	12	3333	30000	2-Mar-2020
2	Bharath	12	8888	35000	2-Mar-2020
3	Chethan	12	5555	25000	2-Mar-2020
4	Dhruva	13	6666	25000	2-Mar-2020
5	Elizer	13	7777	45000	2-Mar-2020

DAY2 Data

SSN	NAME	AGE	PHNO	SAL	Modified Date
6	Faheem	12	8181	60000	3-Mar-2020
7	Guru	14	9999	50000	3-Mar-2020
2	Bharath	12	8888	40000	3-Mar-2020
3	Chethan	12	5555	30000	3-Mar-2020
8	John	13	4040	45000	3-Mar-2020

STAGE

SSN	NAME	AGE	PHNO	SAL	Modified Date
1	Abhilash	12	3333	30000	2-Mar-2020
2	Bharath	12	8888	40000	2-Mar-2020
3	Chethan	12	5555	30000	2-Mar-2020
4	Dhruva	13	6666	25000	2-Mar-2020
5	Elizer	13	7777	45000	2-Mar-2020
6	Faheem	12	8181	60000	3-Mar-2020
7	Guru	14	9999	50000	3-Mar-2020
2	Bharath	12	8888	40000	3-Mar-2020
3	Chethan	12	5555	30000	3-Mar-2020
8	John	13	4040	45000	3-Mar-2020

DATA WAREHOUSE

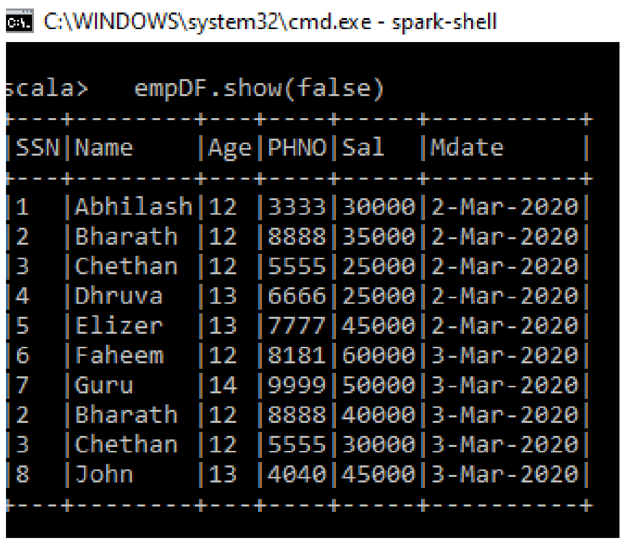
SSN	NAME	AGE	PHNO	SAL	Modified Date
1	Abhilash	12	3333	30000	2-Mar-2020
2	Bharath	12	8888	40000	2-Mar-2020
3	Chethan	12	5555	30000	2-Mar-2020
4	Dhruva	13	6666	25000	2-Mar-2020
5	Elizer	13	7777	45000	2-Mar-2020
6	Faheem	12	8181	60000	3-Mar-2020
7	Guru	14	9999	50000	3-Mar-2020
8	John	13	4040	45000	3-Mar-2020

Fig1: Illustration of CDC

The above techniques is a time stamp based CDC[15][16], in which it's mandatory to have the changed time stamp of each row. The implementation using big data technologies like hadoop , spark and scala was done to improve on the existing technique. The code was written in scala and ETL pipeline was spark. The below code snippet assigns a sequence of structured data to a dataframe[17]. The dataframe name is empDF and has 10 rows i.e. 5 row each for Day1 and Day2.

3.1 DATA FRAMES AS STAGING FOR CDC

Data frame contents having 10 rows is shown in below Fig2. Output of the CDC based on time stamp for two days data is shown in the below Fig3. The output shows only 8 rows out of 10 rows indicating 2 rows that were duplicates have been discarded based on max modified date. So the old 5 rows and newly added 3 rows sum up to 8 rows.



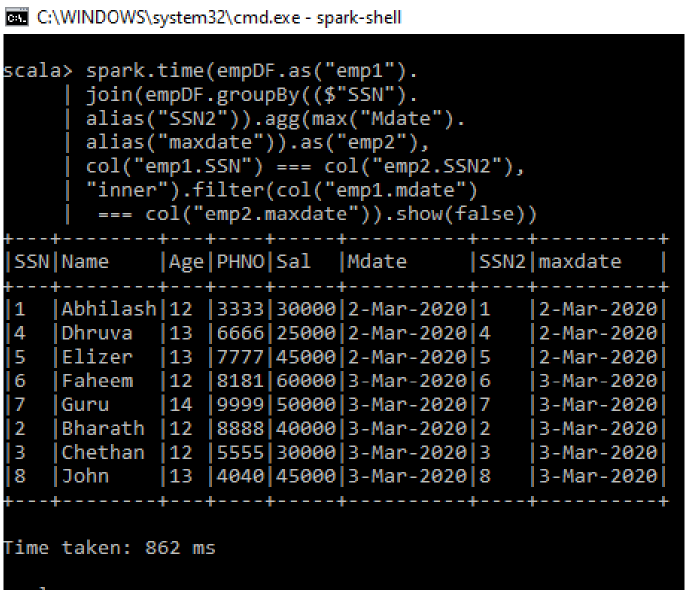
```
C:\WINDOWS\system32\cmd.exe - spark-shell

scala>   empDF.show(false)
+---+--------+---+----+-----+----------+
|SSN|Name    |Age|PHNO|Sal  |Mdate     |
+---+--------+---+----+-----+----------+
|1  |Abhilash|12 |3333|30000|2-Mar-2020|
|2  |Bharath |12 |8888|35000|2-Mar-2020|
|3  |Chethan |12 |5555|25000|2-Mar-2020|
|4  |Dhruva  |13 |6666|25000|2-Mar-2020|
|5  |Elizer  |13 |7777|45000|2-Mar-2020|
|6  |Faheem  |12 |8181|60000|3-Mar-2020|
|7  |Guru    |14 |9999|50000|3-Mar-2020|
|2  |Bharath |12 |8888|40000|3-Mar-2020|
|3  |Chethan |12 |5555|30000|3-Mar-2020|
|8  |John    |13 |4040|45000|3-Mar-2020|
+---+--------+---+----+-----+----------+
```

Fig2: Two days data combined in a single data frame (staging)

Volume 10, Issue 1
February 2021

ITEE Journal
Information Technology & Electrical Engineering

ISSN: - 2306-708X



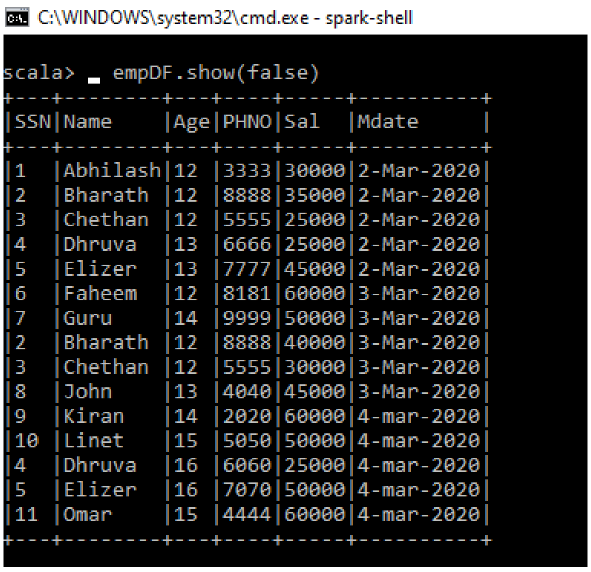
```
C:\WINDOWS\system32\cmd.exe - spark-shell

scala> spark.time(empDF.as("emp1").
     | join(empDF.groupBy(($"SSN").
     | alias("SSN2")).agg(max("Mdate").
     | alias("maxdate")).as("emp2"),
     | col("emp1.SSN") === col("emp2.SSN2"),
     | "inner").filter(col("emp1.mdate")
     |  === col("emp2.maxdate")).show(false))
+---+--------+---+----+-----+----------+----+----------+
|SSN|Name    |Age|PHNO|Sal  |Mdate     |SSN2|maxdate   |
+---+--------+---+----+-----+----------+----+----------+
|1  |Abhilash|12 |3333|30000|2-Mar-2020|1   |2-Mar-2020|
|4  |Dhruva  |13 |6666|25000|2-Mar-2020|4   |2-Mar-2020|
|5  |Elizer  |13 |7777|45000|2-Mar-2020|5   |2-Mar-2020|
|6  |Faheem  |12 |8181|60000|3-Mar-2020|6   |3-Mar-2020|
|7  |Guru    |14 |9999|50000|3-Mar-2020|7   |3-Mar-2020|
|2  |Bharath |12 |8888|40000|3-Mar-2020|2   |3-Mar-2020|
|3  |Chethan |12 |5555|30000|3-Mar-2020|3   |3-Mar-2020|
|8  |John    |13 |4040|45000|3-Mar-2020|8   |3-Mar-2020|
+---+--------+---+----+-----+----------+----+----------+

Time taken: 862 ms
```

Fig3: Two days data self-join CDC

Next Fig4 shows continued data arrival for Day3. Staging table already contains 10 rows from Day1 and Day2 now after combining with Day3 there are 15 rows. After the self-join operation because of repetition of certain rows only the latest modified and inserted rows are retained as shown in Fig5. Day4 data is now being integrated and CDC logic is performed as shown in Fig6and Fig7 respectively. As compared to joining two days data joining third day and fourth day data took a little extra time in milliseconds. So the results show that as the data keeps on arriving it's placed in stage table and CDC is performed. The illustration was made using days as interval for arrival of files. In real time it could be possible that the files arrive at every hour or after certain minutes. If the CDC solution is to put the file content on to the staging area and perform self-joins for getting changed data, then we have identified a problem.

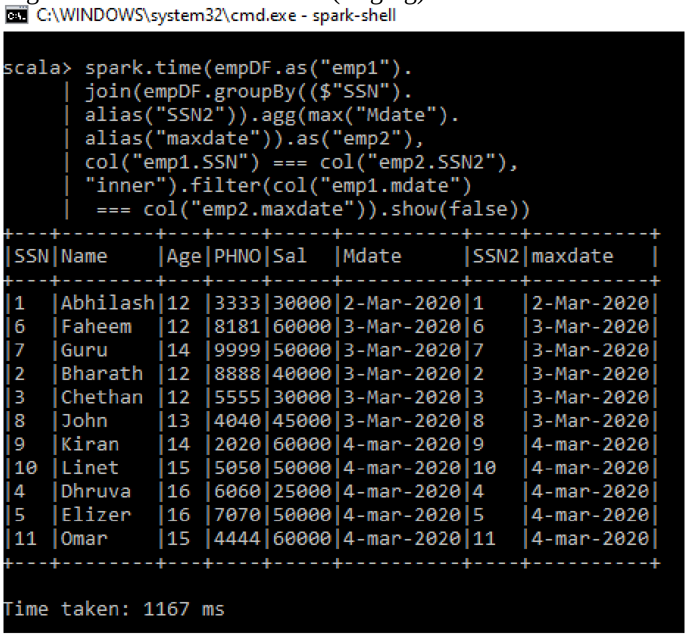
3.2 PROBLEM IDENTIFICATION

The problem is increase in the number of comparisons required every time CDC is computed. The reason of increase in number of comparison is explained as follows. Initially the data in the data frame is 10 rows. Later in the day two times updated files were received and each file needs to be joined with the data frame. Each updates files contains 5 records for illustration purpose. If the first file is update with initial file using a single data frame then it requires 15 multiplied by 15 i.e. 225 comparisons. Later the second file is updated with previous processed 15 records that requires another 20 multiplied by 20 comparisons i.e. 400 comparisons. In total the two files are compared separately with the initial data requires 225 plus 400 i.e. 625 comparisons. The proposed method is to combine the two file contents before placing them into staging area. By this only 20 records are placed in staging area 5 records each from two files. This would reduce the number of comparison because now the staging area contain only 20 records from both the files and only these 20 records needs to be self-joined to get the changed data. Thus by this approach as explained in the previous case instead of comparing 15X15 records and later again comparing 20X20 records only 20X20 records needs to be compared..



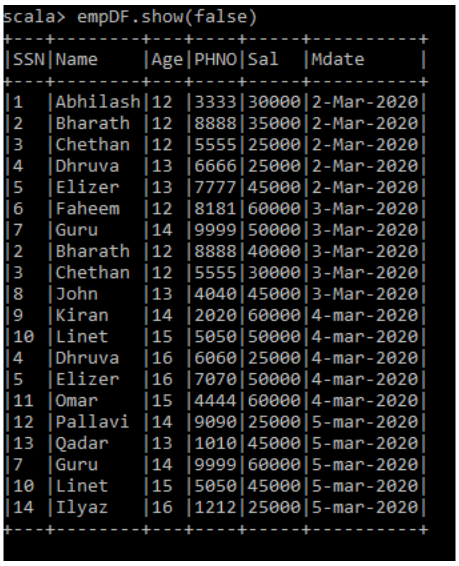
```
C:\WINDOWS\system32\cmd.exe - spark-shell

scala>   empDF.show(false)
+---+--------+---+----+-----+----------+
|SSN|Name    |Age|PHNO|Sal  |Mdate     |
+---+--------+---+----+-----+----------+
|1  |Abhilash|12 |3333|30000|2-Mar-2020|
|2  |Bharath |12 |8888|35000|2-Mar-2020|
|3  |Chethan |12 |5555|25000|2-Mar-2020|
|4  |Dhruva  |13 |6666|25000|2-Mar-2020|
|5  |Elizer  |13 |7777|45000|2-Mar-2020|
|6  |Faheem  |12 |8181|60000|3-Mar-2020|
|7  |Guru    |14 |9999|50000|3-Mar-2020|
|2  |Bharath |12 |8888|40000|3-Mar-2020|
|3  |Chethan |12 |5555|30000|3-Mar-2020|
|8  |John    |13 |4040|45000|3-Mar-2020|
|9  |Kiran   |14 |2020|60000|4-mar-2020|
|10 |Linet   |15 |5050|50000|4-mar-2020|
|4  |Dhruva  |16 |6060|25000|4-mar-2020|
|5  |Elizer  |16 |7070|50000|4-mar-2020|
|11 |Omar    |15 |4444|60000|4-mar-2020|
+---+--------+---+----+-----+----------+
```

Fig4: Three data files 15 rows (staging)



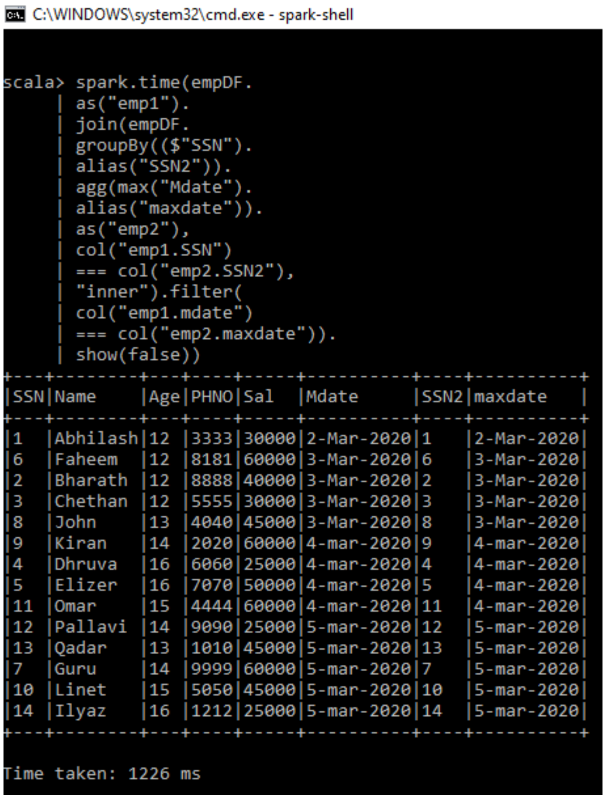
```
C:\WINDOWS\system32\cmd.exe - spark-shell

scala> spark.time(empDF.as("emp1").
     | join(empDF.groupBy(($"SSN").
     | alias("SSN2")).agg(max("Mdate").
     | alias("maxdate")).as("emp2"),
     | col("emp1.SSN") === col("emp2.SSN2"),
     | "inner").filter(col("emp1.mdate")
     |  === col("emp2.maxdate")).show(false))
+---+--------+---+----+-----+----------+----+----------+
|SSN|Name    |Age|PHNO|Sal  |Mdate     |SSN2|maxdate   |
+---+--------+---+----+-----+----------+----+----------+
|1  |Abhilash|12 |3333|30000|2-Mar-2020|1   |2-Mar-2020|
|6  |Faheem  |12 |8181|60000|3-Mar-2020|6   |3-Mar-2020|
|7  |Guru    |14 |9999|50000|3-Mar-2020|7   |3-Mar-2020|
|2  |Bharath |12 |8888|40000|3-Mar-2020|2   |3-Mar-2020|
|3  |Chethan |12 |5555|30000|3-Mar-2020|3   |3-Mar-2020|
|8  |John    |13 |4040|45000|3-Mar-2020|8   |3-Mar-2020|
|9  |Kiran   |14 |2020|60000|4-mar-2020|9   |4-mar-2020|
|10 |Linet   |15 |5050|50000|4-mar-2020|10  |4-mar-2020|
|4  |Dhruva  |16 |6060|25000|4-mar-2020|4   |4-mar-2020|
|5  |Elizer  |16 |7070|50000|4-mar-2020|5   |4-mar-2020|
|11 |Omar    |15 |4444|60000|4-mar-2020|11  |4-mar-2020|
+---+--------+---+----+-----+----------+----+----------+

Time taken: 1167 ms
```

Fig5: Three data files CDC using self-join



```
scala> empDF.show(false)
+---+--------+---+----+-----+----------+
|SSN|Name    |Age|PHNO|Sal  |Mdate     |
+---+--------+---+----+-----+----------+
|1  |Abhilash|12 |3333|30000|2-Mar-2020|
|2  |Bharath |12 |8888|35000|2-Mar-2020|
|3  |Chethan |12 |5555|25000|2-Mar-2020|
|4  |Dhruva  |13 |6666|25000|2-Mar-2020|
|5  |Elizer  |13 |7777|45000|2-Mar-2020|
|6  |Faheem  |12 |8181|60000|3-Mar-2020|
|7  |Guru    |14 |9999|50000|3-Mar-2020|
|2  |Bharath |12 |8888|40000|3-Mar-2020|
|3  |Chethan |12 |5555|30000|3-Mar-2020|
|8  |John    |13 |4040|45000|3-Mar-2020|
|9  |Kiran   |14 |2020|60000|4-mar-2020|
|10 |Linet   |15 |5050|50000|4-mar-2020|
|4  |Dhruva  |16 |6060|25000|4-mar-2020|
|5  |Elizer  |16 |7070|50000|4-mar-2020|
|11 |Omar    |15 |4444|60000|4-mar-2020|
|12 |Pallavi |14 |9090|25000|5-mar-2020|
|13 |Qadar   |13 |1010|45000|5-mar-2020|
|7  |Guru    |14 |9999|60000|5-mar-2020|
|10 |Linet   |15 |5050|45000|5-mar-2020|
|14 |Ilyaz   |16 |1212|25000|5-mar-2020|
+---+--------+---+----+-----+----------+
```

Fig6: Four data files 20 rows



C:\WINDOWS\system32\cmd.exe - spark-shell

```
scala> spark.time(empDF.
     | as("emp1").
     | join(empDF.
     | groupBy(($"SSN").
     | alias("SSN2")).
     | agg(max("Mdate").
     | alias("maxdate")).
     | as("emp2"),
     | col("emp1.SSN")
     | === col("emp2.SSN2"),
     | "inner").filter(
     | col("emp1.mdate")
     | === col("emp2.maxdate")).
     | show(false))
+---+--------+---+----+-----+----------+----+----------+
|SSN|Name    |Age|PHNO|Sal  |Mdate     |SSN2|maxdate   |
+---+--------+---+----+-----+----------+----+----------+
|1  |Abhilash|12 |3333|30000|2-Mar-2020|1   |2-Mar-2020|
|6  |Faheem  |12 |8181|60000|3-Mar-2020|6   |3-Mar-2020|
|2  |Bharath |12 |8888|40000|3-Mar-2020|2   |3-Mar-2020|
|3  |Chethan |12 |5555|30000|3-Mar-2020|3   |3-Mar-2020|
|8  |John    |13 |4040|45000|3-Mar-2020|8   |3-Mar-2020|
|9  |Kiran   |14 |2020|60000|4-mar-2020|9   |4-mar-2020|
|4  |Dhruva  |16 |6060|25000|4-mar-2020|4   |4-mar-2020|
|5  |Elizer  |16 |7070|50000|4-mar-2020|5   |4-mar-2020|
|11 |Omar    |15 |4444|60000|4-mar-2020|11  |4-mar-2020|
|12 |Pallavi |14 |9090|25000|5-mar-2020|12  |5-mar-2020|
|13 |Qadar   |13 |1010|45000|5-mar-2020|13  |5-mar-2020|
|7  |Guru    |14 |9999|60000|5-mar-2020|7   |5-mar-2020|
|10 |Linet   |15 |5050|45000|5-mar-2020|10  |5-mar-2020|
|14 |Ilyaz   |16 |1212|25000|5-mar-2020|14  |5-mar-2020|
+---+--------+---+----+-----+----------+----+----------+

Time taken: 1226 ms
```

Fig7: Comparing proposed approach with previous approach

3.3 PROPOSED SOLUTION

Our claim is supported by the experimental results shown in Fig5 and Fig7. Three file combined in staging area contains 15 records as shown in Fig4 (files 3 contains 5 records and integrating it with the staging data containing 10 rows) and perform CDC takes 1167ms. Again after the arrival of file4 again containing 5 records and integrating it with the stating data and performing CDC takes 1226ms. The approach proposed is if there is a little difference in time of arrival of file3 and file4 the entire CDC could be done in 1226ms instead of integrating and performing CDC on file3 and file4 separately. But this has to be done if the interval of arrival of file is less than the total time taken for performing two separate CDC operations. If the interval of arrival is reduced then because of requirement of real time analytics separate CDC needs to be done.

4. IMPLEMENTATION

Approach to combine data into staging before performing CDC was implemented on TPC-DS [18]bench mark dataset. The CDC is performed on dimension table customer. The size of data was 1GB and out of this customer dimension contained 100k rows. Machine used was intel i5 core with 4GB RAM. Because of the limited hardware distributed cluster was simulated using winutils[19]. Data was experimented with initial load of 10k and 50k. Each of the initial load were further incrementally loaded with percentage of loads ranging from 10%, 20%, 30%, 50% and 80%. These percentage loads were performed for both data present in combined vs separate files. An example combined of 10% incremental load for the initial load of 10k would require 1k new data entries. Further this 1k is split into approximately 80% new rows and 20% modified rows. Similarly an example of separate 10% would require two files and data split of exactly 50% in each file. Hence the separate files would contain 500 rows each and percentage of newly added and modified rows could be again approximately 80% and 20%.

5. RESULTS AND DISCUSSION

The results as shown in Fig8 and Fig9 of the two approaches are plotted by comparing the time required to perform CDC. In all cases it's been observed that combined approach is better in comparison to separate file based approach. Also a plot of difference in time between the two approaches for performing CDC for various loads is shown in Fig10. It proves that as the data volumes increase and the percentage of CDC data increases the difference increases and hence combined approach is better off compared to having data in separate files.

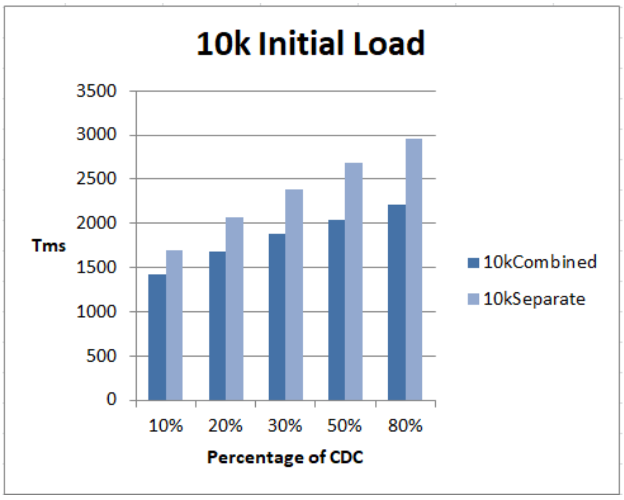


Fig8: Comparison of CDC for 10k initial load

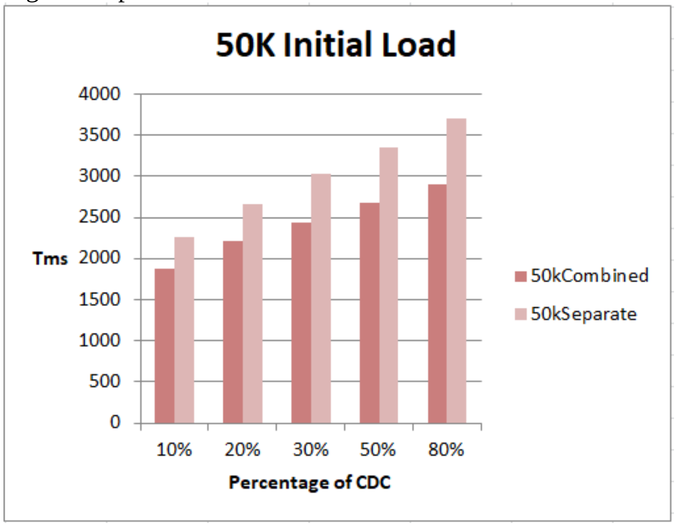


Fig9: Comparison of CDC for 50k initial load

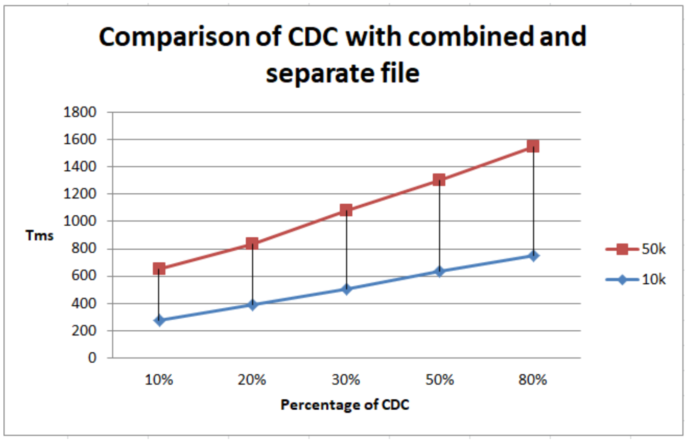


Fig10: Difference in time for various loads percentages

6. CONCLUSION AND FUTURE ENHANCEMENT

Maintenance of data warehouse projects is a very critical component for decision support system because without updated data analysis could not yield the desired results. Part of maintenance is the process to identify changes from the source and move to the data warehouse. In this work we have implemented a multi-stage CDC that helps in reducing the efficiency of process to identify updated data from the source system. The empirical study shows that when two or more files arriving from source combined into a staging area before actually performing the task of CDC is more efficient compared to handling every single file individually at a single staging area. Also because this implementation was on spark based ETL engine there is one more problem .i.e. it takes extra time for opening and closing of files if the number of files are more, as compared to opening and closing combined file once. We have not taken into account in this work the opening and closing time in this work but only reduced the number of comparison and thereby increasing the efficiency. In future we plan to implement a CDC scheme on a multi-node cluster using cloud services like Google cloud platform or amazon s3. Also there is a lot of scope in experimentation and exploration of techniques to identify changed data in semi-structured and un-structured sources. We plan to implement and study CDC for semi-structured sources in future.

REFERENCES

[1] C.E. Offia, M. Crowe, a Theoritical Exploration of Data Management and Integration in Organisation Sectors, Int. J. Database Manag. Syst. 11 (2019) 37–56. https://doi.org/10.5121/ijdms.2019.11103.

[2] J. Dean, S. Ghemawat, MapReduce: Simplified data processing on large clusters, Commun. ACM. 51 (2008) 107–113. https://doi.org/10.1145/1327452.1327492.

[3] X. Liu, C. Thomsen, T.B. Pedersen, Mapreduce-based dimensional ETL made easy, Proc. VLDB Endow. 5 (2012) 1882–1885. https://doi.org/10.14778/2367502.2367528.

[4] R. Sherman, Change data capture, in: R. Sherman (Ed.), Bus. Intell. Guideb., Morgan Kaufmann, 2015: pp. 301–333. https://doi.org/10.1145/1723028.1723064.

[5] X. Liu, C. Thomsen, T.B. Pedersen, ETLMR: A highly scalable dimensional ETL framework based on MapReduce, Lect. Notes Comput. Sci. (Including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics). 7790 LNCS (2013) 1–31. https://doi.org/10.1007/978-3-642-37574-3_1.

[6] X. Liu, C. Thomsen, T.B. Pedersen, CloudETL:

Scalable dimensional ETL for hive, ACM Int. Conf. Proceeding Ser. (2014) 195–206. https://doi.org/10.1145/2628194.2628249.

[7] M. Bala, O. Boussaid, Z. Alimazighi, Big-ETL: extracting-transforming-loading approach for Big Data, Proc. Int. Conf. Parallel Distrib. Process. Tech. Appl. (2015) 462.

[8] X. Liu, N. Iftikhar, An ETL optimization framework using partitioning and parallelization, Proc. ACM Symp. Appl. Comput. 13-17-Apri (2015) 1015–1022. https://doi.org/10.1145/2695664.2695846.

[9] M. Bala, O. Boussaid, Z. Alimazighi, Extracting-transforming-loading modeling approach for big data analytics, Int. J. Decis. Support Syst. Technol. 8 (2016) 50–69. https://doi.org/10.4018/IJDSST.2016100104.

[10] P.M. B, M. Abbasi, P. Furtado, AScale: Big/Small Data ETL and Real-Time Data Freshness, (2016) 315–327. https://doi.org/10.1007/978-3-319-34099-9.

[11] X. Liu, N. Iftikhar, H. Huo, P.S. Nielsen, Optimizing ETL by a two-level data staging method, Int. J. Data Warehous. Min. 12 (2016) 32–50. https://doi.org/10.4018/IJDWM.2016070103.

[12] M. Barkhordari, M. Niamanesh, Chabok: a Map-Reduce based method to solve data warehouse problems, J. Big Data. 5 (2018). https://doi.org/10.1186/s40537-018-0144-5.

[13] Denny, I.P.M. Atmaja, A. Saptawijaya, S. Aminah, Implementation of change data capture in ETL process for data warehouse using HDFS and apache spark, Proc. - WBIS 2017 2017 Int. Work. Big Data Inf. Secur. 2018-Janua (2018) 49–55. https://doi.org/10.1109/IWBIS.2017.8275102.

[14] R. Michał Bodziony, Szymon, Roszyk, Wrembel, On Evaluating Performance of Balanced Optimization of ETL Processes for Streaming Data Sources, in: DOLAP, 2020: pp. 2–6.

[15] Incremental Data Load in Hive, 2019. https://www.youtube.com/watch?v=B51yDF04xLw.

[16] Talend, Change Data Capture, n.d. https://help.talend.com/r/5csctXNxxEsUTzH9FeCmmg/jN_3DNv5O4ShiIZtSBDMKA.

[17] Prashanth, Apache Spark Foundation Course - Dataframe Basics, Learn. Journals. (2018). https://www.learningjournal.guru/courses/spark/spark-foundation-training/spark-dataframe-basics/.

[18] Transaction Processing Council, (2020) http://www.tpc.org/tpc_documents_current_versions/. http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-ds_v2.13.0.pdf.

[19] Steveloughran, Windows binaries for Hadoop versions, (n.d.). https://github.com/steveloughran/winutils.

AUTHOR PROFILES

Mohammed Muddasir N is currently working as a assistant professor in the department of Information Science and Engineering, Vidyavardhaka College of Engineering, Mysuru, India. He is having teaching experience of 8 years and industrial experience of 4 years. He is currently a research scholar with Visvesvaraya Technological University, Belagavi, India. He has a bachelor's degree in computer science and master's degrees in network and internet.

Raghuveer K has been at the National Institute of Engineering since 1994. He has completed B.E (Computer Science & Engineering) from University of Mysore in 1988, M.E from Devi Ahilya Vishwavidyalaya, formerly known University of Indore in 1993 and PhD in 2008. He has worked as Head, Department of Information Science & Engineering from 2008 to 2019. He has served as Head, Internal Quality Assurance Cell (IQAC).Currently he is serving as Principal of National Institute of Engineering.