Advances in Science, Technology and Engineering Systems Journal Vol. 4, No. 5, 327-333 (2019)

www.astesj.com

Special Issue on Advancement in Engineering and Computer Science

ASTESJ

ISSN: 2415-6698

Improving the Performance of Hadoop Framework Using Optimization Process in the Information Management

Ramachandran Ravi Sowmiyasree, Nachimuthu Maheswari*, Manickam Sivagami

School of Computing Science and Engineering, Vellore Institute of Technology, Chennai, India

A R T I C L E I N F O

Article history:
Received: 01 June, 2019
Accepted: 17 September, 2019
Online: 22 October, 2019

Keywords:
Bigdata
Hadoop
Performance
Map reduce
Hive
sqoop

A B S T R A C T

Hadoop has certain issues that could be taken care to execute the job efficiently. These limitations are due to the locality of the data in the cluster, allocation of the jobs, scheduling of the tasks and resource allocations in Hadoop. Execution in the mapreduce remains a challenge in terms of efficiency. So, an improved Hadoop architecture that takes care of the computation time has been discussed. The improved architecture addresses the communication issues with the task trackers, inefficient clean up task, heartbeat function. Comparing with native Hadoop, the improved Hadoop reduces the total time taken for running the reducer tasks. The performance of the improved system using optimization serves better in the computation time.

1. Introduction

Big data has a huge volume, velocity and variety of information, which produces cost effective, new forms of information processing that helps in decision making, prediction and automation etc. Big data analytics is the process of analyzing large and varied data sets i.e., big data to discover hidden patterns, unknown correlations, trends, user preferences and other useful information that can help organizations to take better decisions in business. Parallel processing in data analytics has appeared as an interdisciplinary research area due to the nature and large size of data [1]. Pattern matching/mining or analysis need huge amount of complex data processing and computing [2].

Significant and reasonable allocation of resources are required to solve complicated problems [3]. It is very difficult to understand and process the data using traditional processing techniques. Big data parallel processing platforms has selected new possibilities to process the structured, semi-structured or unstructured data [4].

1.1 Problem Statement

Data localization and the resource allocation for the tasks are the challenges in the Hadoop. Effective and efficient resource allocation are the challenges in the map reduce framework. In order to implement an architecture with the data gathered from different sources of the systems as in the banking practice and various data processing supports as in the traditional global financial systems, an improvement in the computation time is required.

1.2 Motivation

Financial sectors and the banks are facing severe demands due to the growth of data processing. This happens not only from the improved regulatory requirements and an inconsistency in the data sources. The cost has to be reduced without compromising scalability and flexibility. In this scenario, the financial services industry shows tremendous interest in applying big data technologies to extract the significant value from the large amount of generated data. So, there is a requirement to improve the computation time of the process.

The proposed system addresses the issues in communicating with the task trackers, inefficient cleanup tasks, heartbeat function. Time taken for the reducers to complete the task has been reduced, which helps in better computation time.

The sections of the paper are divided as follows: Section 2 discussed some related works. In section 3, Hadoop map reduce

* Nachimuthu Maheswari, Email: maheswari.n@vit.ac.in

work flow and its performance is discussed. In section 4, the enhanced hadoop system is discussed. In section 5, the implementation phase is discussed. In section 6, the results are evaluated and discussed. In section 7, the conclusion is provided. Finally, in section 8, the future works are discussed.

2. Related Work

The improvement in Hadoop Mapreduce performance has suggested from different concerns or aspects. Many works have suggested in the improvement of the performance of mapreduce jobs and the hadoop, such as scheduling the jobs and computation time improvement. Scheduling the job and the execution time are considered as the important features of Hadoop [5] [6]. Various studies have given the information and have received the results using their ideas [7] [8]. Few people focus on the time of starting and ending process of map reduce job processing [9]. Issues in the system memory could be addressed to improve the performance of the overall system [10]. Apart from Hadoop, few studies utilize a distributed caching method to enhance the performance of Hadoop [11] [12]. Shm Streaming [13] proposes a streaming type of schema to give First In First Out queue as lock less which joins Hadoop and other programs. Hadoop spreads the redundant data into various nodes in various racks. This will be helpful in false tolerant issues. Various studies consider the improvements in data locality development for the betterment of hadoop [14]. Few, discuss on the data types for the betterment in the performance of Hadoop [15].

3. Hadoop Overview

Hadoop is an open source framework for distributed storage and processing. It gives solutions for big data processing and analysis. Hadoop has a file system that stores the data relevant to the applications. The interface is known as the Hadoop Distributed File System (HDFS). Hadoop Distributed File System shares the resources for data analysis. The major parts of Hadoop are map reduce and Hadoop Distributed File System. Other hand, moving the computing towards the data is less costlier than movement of data towards computing [16]. Hadoop allocates file system to store huge amount of data files across the nodes in the clusters.

In HDFS, the Hadoop cluster has two components, which are the masters- Name Node and the slaves- Data Nodes. In Hadoop cluster, name node is responsible for maintaining the file system. It helps in maintaining the data and sending the jobs to the corresponding data nodes responsible for the application data [17]. Job tracker works in the master node. Job Tracker takes care of the data related to the applications in the data nodes with the help of the task tracker for processing. Each task is running in the respective allotted slot in a data node, which has a fixed amount of map/reduce slots.

3.1 Mapreduce

Map reduce processes the job that splits the input data of the job into independent parts and stores the data in HDFS. During map reduce execution phase, multiple map tasks are processed in parallel. After the completion of the map tasks multiple reduce tasks are processed in parallel [18]. With respect to the applications the total number of allotted map tasks be different than that of reduce tasks. Data that are stored in HDFS and processed in map/reduce framework is the form of key and value pair. It has stored and used in the map/reduce tasks to determine the required results at the end of the job. The final results will be displayed in the key and value combination.

3.2 Mapreduce Work flow

HDFS stored the data required for the map reduce job and the data is distributed in to the blocks. Each mapper processed one block in the same time. Inside the mapper phase, the user can give the logic as per theproblem requirements. So that the map tasks runs on all the nodes in the cluster and computes the data parallelly that are stored in the blocks.

The mapper output is stored in the local disk as it is only the intermediate output. If it is written on HDFS, that will create unnecessary multiple copies as the HDFS always replicates the data. The mapper output is sorted/shuffled and submitted to the reducer. The shuffling/sorting has taken care by the internal process. Reducer phase is the next phase of processing and the user can specify his/her own statements. Input to the reducer is given by all the mappers. The final output is produced by the reducer and written on HDFS.

4. Improved Hadoop System

The focus is on transferring the relational data into HDFS using sqoop. The exported data is analysed using Hive by using Hive Query Language [19]. The data is partitioned and is fed into MapReduce for further processing. The MapReduce jobs are optimized due to the changes done in the configuration of Job in progress, Task in progress and Task tracker files of hadoop.

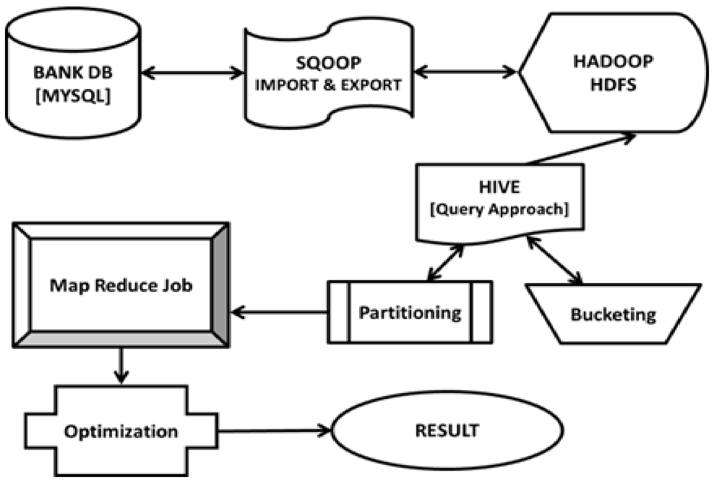


Figure.1 Diagrammatic representation of the proposed system

The bank MySql Database contains the required datasets. The data is exported to HDFS using Sqoop tool. Then Hive does the partition of the exported data, for which the results gets stored in HDFS. Then the Hive partitioned data is analysed and fetched from HDFS for MapReduce processing. The next step is Optimisation, which is like the heart of the work. Once the optimisation is over, the results are compared with that of the native hadoop to prove the working correctness of the work as shown in Figure 1. The flow of the proposed system is explained below:

System Work flow

Step 1: Client requests for data sets. Bank data set is transferred from Mysql to HDFS using Sqoop.

Step 2: Hive extracts the data and partition the data set for map reduce programming.

Step 3: Changes performed in the Jobinprogress, Taskin progress, task tracker files.

Step 4: Name node initializes the job. Data fetched from the data node and performs the job.

Step 5: Performance of the results with respect to the computation time is compared with the native Hadoop.

4.1 Data Collection and Preprocessing

The Initial phase of the work is data collection. The article has a bank dataset that contains the account details, customer details and transaction details in MySql database.The transfer of the dataset into hadoop (HDFS), i.e, data migration is done using Sqoop tool. Sqoop supports for transferring data from relational databases to Hadoop.

Data from the three datasets (account,customer and transaction) are combined with its attributed and produced 21,215 records. The final combined dataset has the following attributes: account number, transaction type, amount, month and year.

In this phase, the dataset is fetched into hadoop (HDFS) using Sqoop Tool as in Figure 2. Using Sqoop, customized functionalities can be performed like fetching the particular column or fetching the dataset with specific condition and data can be stored in hadoop (HDFS).

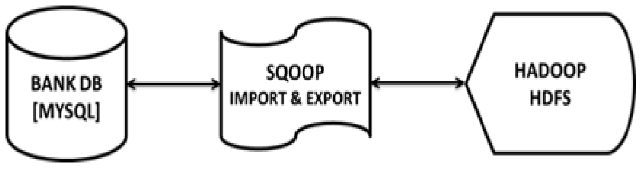


Figure.2 Data preprocessing using sqoop

4.2 Data Analysis

In this phase, the dataset is analysed using HIVE tool which will be stored in hadoop (HDFS). For analyzing the dataset, HIVE uses HQL Language. Hive is chosen because it is a data warehouse by itself. Hive has the ability to work on top of an existing Hadoop cluster and provides SQL-like interface. The user can map the existing Sqoop tables to Hive and operate on them as shown in Figure3.

4.2.1 Partitioning

Hive arranges the tables into partitions. The records are partitioned based on the transaction type(with draw, transfer, deposit). Partitioning helps in dividing a table into related parts based on the values of partitioned columns as shown in Figure 4. Partitioning is useful in querying a portion of the data.

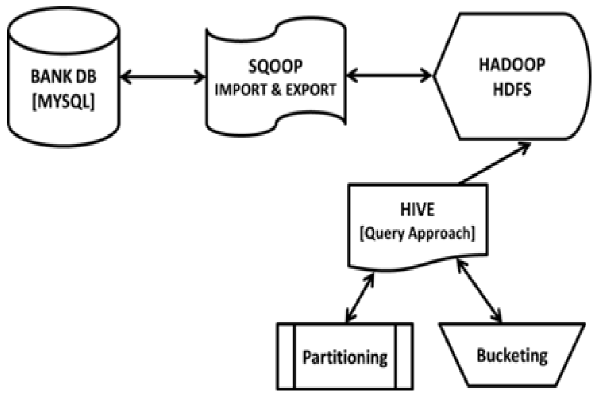
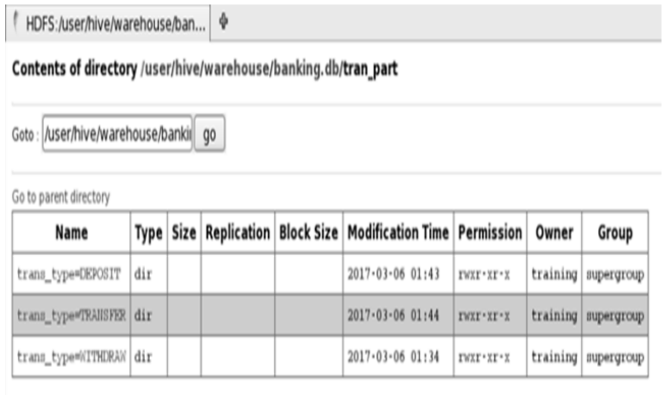


Figure.3 Data analysis using Hive



Contents of directory /user/hive/warehouse/banking.db/**tran_part**

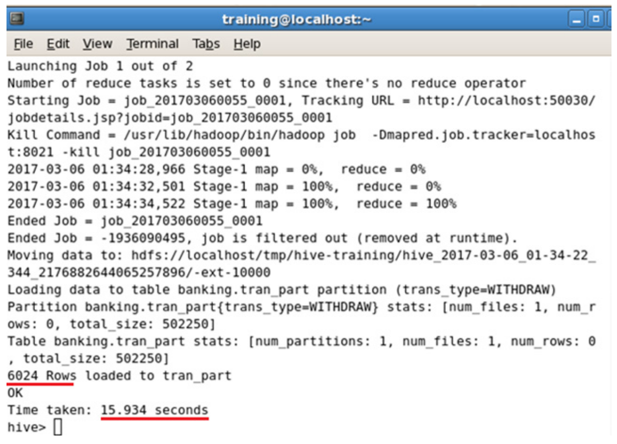
Goto : /user/hive/warehouse/banki go

Go to parent directory

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
trans_type=DEPOSIT	dir				2017-03-06 01:43	rwxr-xr-x	training	supergroup
trans_type=TRANSFER	dir				2017-03-06 01:44	rwxr-xr-x	training	supergroup
trans_type=WITHDRAW	dir				2017-03-06 01:34	rwxr-xr-x	training	supergroup

Figure.4 Partitioning the transaction data

The number of rows loaded and the time taken to load the partitioned data into HDFS is shown in Figure 5.



```
Launching Job 1 out of 2
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201703060055_0001, Tracking URL = http://localhost:50030/
jobdetails.jsp?jobid=job_201703060055_0001
Kill Command = /usr/lib/hadoop/bin/hadoop job  -Dmapred.job.tracker=localhos
t:8021 -kill job_201703060055_0001
2017-03-06 01:34:28,966 Stage-1 map = 0%,  reduce = 0%
2017-03-06 01:34:32,501 Stage-1 map = 100%,  reduce = 0%
2017-03-06 01:34:34,522 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_201703060055_0001
Ended Job = -1936090495, job is filtered out (removed at runtime).
Moving data to: hdfs://localhost/tmp/hive-training/hive_2017-03-06_01-34-22_
344_2176882644065257896/-ext-10000
Loading data to table banking.tran_part partition (trans_type=WITHDRAW)
Partition banking.tran_part{trans_type=WITHDRAW} stats: [num_files: 1, num_r
ows: 0, total_size: 502250]
Table banking.tran_part stats: [num_partitions: 1, num_files: 1, num_rows: 0
, total_size: 502250]
6024 Rows loaded to tran_part
OK
Time taken: 15.934 seconds
hive> 
```

Figure.5 Loading partitioned data into HDFS

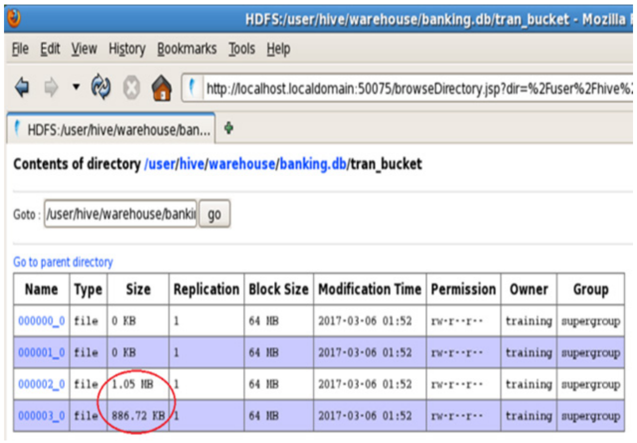
4.2.2 Bucketing

Bucketing decomposes data into manageable or equal parts as shown in Figure 6. The user can restrict the number of buckets to store the data. It provides faster query response. The number of required buckets can be given during the table creation. Loading of equal volume of data has to be done manually by programmers is a significant issue.

Choosing partitioned data over bucketed data

The size of the loaded partitioned data in the three partitions is less when compared to the size of the loaded bucketed data.

Bucketing works well when the field has high cardinality (number of possible values a field can have) and data is evenly distributed among buckets. Partitioning works best when the cardinality of the partitioning field is low.


HDFS:/user/hive/warehouse/banking.db/tran_bucket - Mozilla
File Edit View History Bookmarks Tools Help
http://localhost.localdomain:50075/browseDirectory.jsp?dir=%2Fuser%2Fhive%
HDFS:/user/hive/warehouse/ban...
Goto : /user/hive/warehouse/banki
go
Go to parent directory

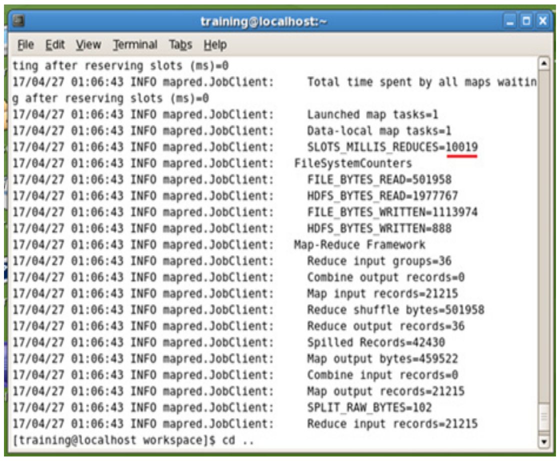
Contents of directory /user/hive/warehouse/banking.db/tran_bucket

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
000000_0	file	0 KB	1	64 MB	2017-03-06 01:52	rw-r--r--	training	supergroup
000001_0	file	0 KB	1	64 MB	2017-03-06 01:52	rw-r--r--	training	supergroup
000002_0	file	1.05 MB	1	64 MB	2017-03-06 01:52	rw-r--r--	training	supergroup
000003_0	file	886.72 KB	1	64 MB	2017-03-06 01:52	rw-r--r--	training	supergroup

Figure.6 Bucketed data in HDFS

4.3 Data Processing

Data Processing is done using Hadoop Madpreduce. The partitioned data is fed into MapReduce for the processing. The main factor considered is the SLOTS_MILLIS_REDUCES[20]. It represents the total time taken for running the reduce tasks in milliseconds in the occupied slots. It also includes the tasks that where started speculatively as shown in Figure 7.



```
ting after reserving slots (ms)=0
17/04/27 01:06:43 INFO mapred.JobClient:     Total time spent by all maps waitin
g after reserving slots (ms)=0
17/04/27 01:06:43 INFO mapred.JobClient:     Launched map tasks=1
17/04/27 01:06:43 INFO mapred.JobClient:     Data-local map tasks=1
17/04/27 01:06:43 INFO mapred.JobClient:     SLOTS_MILLIS_REDUCES=10019
17/04/27 01:06:43 INFO mapred.JobClient:   FileSystemCounters
17/04/27 01:06:43 INFO mapred.JobClient:     FILE_BYTES_READ=501958
17/04/27 01:06:43 INFO mapred.JobClient:     HDFS_BYTES_READ=1977767
17/04/27 01:06:43 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=1113974
17/04/27 01:06:43 INFO mapred.JobClient:     HDFS_BYTES_WRITTEN=888
17/04/27 01:06:43 INFO mapred.JobClient:   Map-Reduce Framework
17/04/27 01:06:43 INFO mapred.JobClient:     Reduce input groups=36
17/04/27 01:06:43 INFO mapred.JobClient:     Combine output records=0
17/04/27 01:06:43 INFO mapred.JobClient:     Map input records=21215
17/04/27 01:06:43 INFO mapred.JobClient:     Reduce shuffle bytes=501958
17/04/27 01:06:43 INFO mapred.JobClient:     Reduce output records=36
17/04/27 01:06:43 INFO mapred.JobClient:     Spilled Records=42430
17/04/27 01:06:43 INFO mapred.JobClient:     Map output bytes=459522
17/04/27 01:06:43 INFO mapred.JobClient:     Combine input records=0
17/04/27 01:06:43 INFO mapred.JobClient:     Map output records=21215
17/04/27 01:06:43 INFO mapred.JobClient:     SPLIT_RAW_BYTES=102
17/04/27 01:06:43 INFO mapred.JobClient:     Reduce input records=21215
[training@localhost workspace]$ cd ..
```

Figure.7 Running mapreduce job

4.4 Optimisation

Optimisation can be done to the native hadoop by changing the configuration factors of the system. There are three main files in hadoop[21], to which the changes are done to obtain an improved and an optimised version of hadoop. The files in hadoop to which the changes are made in order to obtain an improved version are

JobInProgress.java

JobInProgress maintains all the information in order to keep the job on the straight. It maintains its JobProfile and its updated JobStatus, along with a set of tables for doing bookkeeping of its tasks.

TaskInProgress.java

TaskInProgress (TIP) maintains all the information required for a task in the lifetime of its owning Job. A given task might be executed or re-executed, so level of indirection is needed above the running-id itself. A given TaskInProgress contains multiple taskids, zero or more of which might be executing at any one time, allowing speculative execution. A TIP allocates enough taskids to account for all the speculation and failures it will ever have to handle. TIP is dead when they are up.

TaskTracker.java

Task tracker is a process that initiates and monitors the map reduce tasks in the environment. It communicates the Job tracker for assigning the tasks and update the status.

4.4.1 JobInProgress

The changes made to the JobInProgress.java are as follows,

Contacting the tasktracker at once

In native hadoop, the task can contact the TT only after the whole task gets completed. So change is made so that, once the runJobSetupTask() works, the task can contact the TT then and there. It is beneficial as the TT is well informed of the on-going tasks and can also help to avoid speculations.

Launching Synchronised Cleanup

In native hadoop, Cleanup task is called once at the end of the task. The change made here is the introduction of Synchronised Cleanup task, which is launched before the cleanup task. It updates the last known cluster size then and there. It is highly beneficial in case of multiple tasks.

Later check of Cleanup

It checks whether the cleanup task is launched already or if the setup is not launched already. The later check is beneficial when the number of maps is zero.

4.4.2 TaskInProgress

The changes made to the TaskInProgress.java are as follows,

Skipping feature

Since completed reduces (for which the outputs go to hdfs) are not failed, this failure is noted only for completed maps, only if the particular tasked completes the particular map. However if the job is done, there is no need to manipulate completed maps. Reset the successful TaskId since no successful tasks can be concluded while running. There can be failures of tasks that are hosted on a machine that has not yet registered with restarted jobtracker. Therefore, the skipping feature recalculates the counts only if it is a genuine failure.

SetComplete function

When the TIP is complete, the other speculative subtasks will be closed when the owning tasktracker reports in and calls should close on the required object.

4.4.3 TaskTracker

The changes made to the TaskTracker.java are as follows,

Resetting heartbeat interval from the response

The name node and the data node communicate using Heartbeat messages. After a certain amount of time, if the Name Node does not receive any response from Data Node, then that particular Data Node is declared as dead. The normal heartbeat period is three seconds. If the name node is not receiving the heartbeat from a data node in ten minutes, then the name node decides the data node to be not in service and the data node's replicas are considered to be unavailable. The name node allocates the new replicas for the unavailable blocks in the other data nodes. The heartbeat message from the data node carries the information regarding the storage, amount of usage and transfer rate of data in the current state. Such details are helpful for the name nodes allocation of blocks and the decision making in load balancing.

By resetting heartbeat interval from the response, the TT knows the completed tasks. Once cleanup of the completed task is over, normal operation is resumed.

5. Implementation

5.1 Mapper Class

The map task is to process the input data. The input data is stored in the Hadoop Distributed File System (HDFS). The input file is passed to the map function and read line by line. The mapper processes the data and produces the intermediate data. The amount, transaction type, year and month are initialised according to the positions in the input data. The mapper class writes the year, month, transaction type and amount along with the sum into HDFS.

5.2 Reducer Class

This stage is the combination of the shuffle and the reduce stage. The Reducers job is to process the data that comes from the mapper. After processing, it produces a new collection of output, which will be stored in the HDFS.

5.3 Driver Class

The Driver class verifies the arguments from the command line. The input/output file details, job details are available in the command line. It assigns values for the job. The driver starts the execution from the main () method. In this method, a new configuration object and the Job are instantiated.

The job.waitForCompletion() helps to unveils the job perfectly. This driver code helps to wait for the job completion. The job status will be updated after the job completion. The true argument informs the framework to write verbose output to the controlling terminal of the job.

6. Results and Discussion

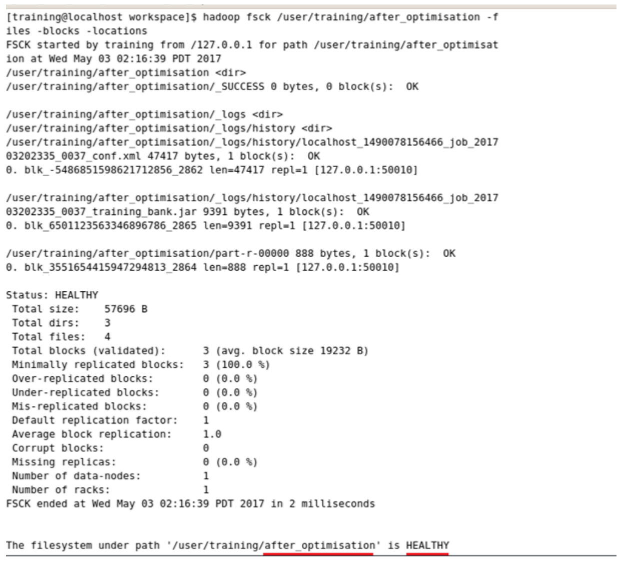
The work has done using bank data, which is analysed to identify the future trends in the sector. The improvement in the Hadoop performance is ensured by the factors such as computation time, time taken for running reduce tasks, contacting the task trackers at once, synchronised clean up, skipping feature, resetting the heartbeat interval.

6.1 Health of the file in HDFS after optimization

The fsck command in hadoop runs a HDFS file system checking utility. It is designed for reporting problems with various files. It checks health of hadoop file system. It produces a report that gives the complete health of the file system. HDFS is considered healthy if all the files have a minimum number of replicas as shown in Figure.8.

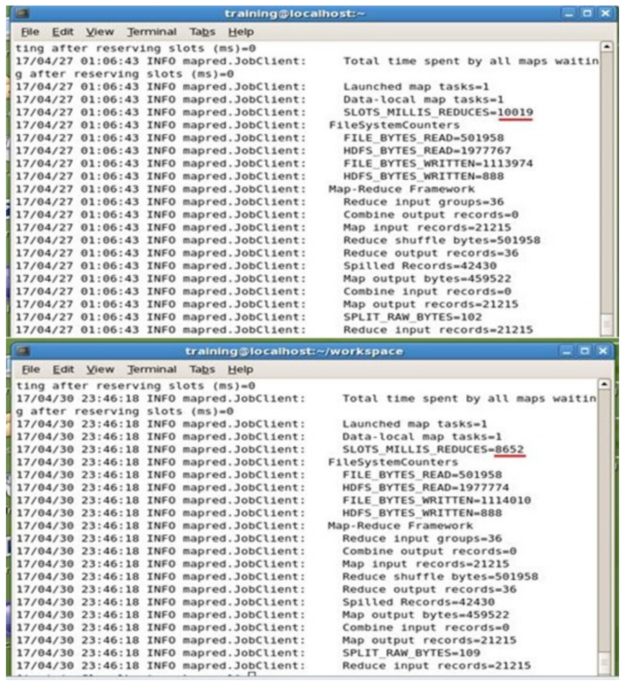
Table 1. Performance Analysis

Factors	Before Optimization(sec)	After Optimization(sec)
Job Duration	82	23
Map Time	60	3
Reduce Time	22	20



```
[training@localhost workspace]$ hadoop fsck /user/training/after_optimisation -f
iles -blocks -locations
FSCK started by training from /127.0.0.1 for path /user/training/after_optimisat
ion at Wed May 03 02:16:39 PDT 2017
/user/training/after_optimisation <dir>
/user/training/after_optimisation/_SUCCESS 0 bytes, 0 block(s):  OK

/user/training/after_optimisation/_logs <dir>
/user/training/after_optimisation/_logs/history <dir>
/user/training/after_optimisation/_logs/history/localhost_1490078156466_job_2017
03202335_0037_conf.xml 47417 bytes, 1 block(s):  OK
0. blk_-5486851598621712856_2862 len=47417 repl=1 [127.0.0.1:50010]

/user/training/after_optimisation/_logs/history/localhost_1490078156466_job_2017
03202335_0037_training_bank.jar 9391 bytes, 1 block(s):  OK
0. blk_6501123563346896786_2865 len=9391 repl=1 [127.0.0.1:50010]

/user/training/after_optimisation/part-r-00000 888 bytes, 1 block(s):  OK
0. blk_3551654415947294813_2864 len=888 repl=1 [127.0.0.1:50010]

Status: HEALTHY
 Total size:    57696 B
 Total dirs:    3
 Total files:   4
 Total blocks (validated):      3 (avg. block size 19232 B)
 Minimally replicated blocks:   3 (100.0 %)
 Over-replicated blocks:        0 (0.0 %)
 Under-replicated blocks:       0 (0.0 %)
 Mis-replicated blocks:         0 (0.0 %)
 Default replication factor:    1
 Average block replication:     1.0
 Corrupt blocks:                0
 Missing replicas:              0 (0.0 %)
 Number of data-nodes:          1
 Number of racks:               1
FSCK ended at Wed May 03 02:16:39 PDT 2017 in 2 milliseconds


The filesystem under path '/user/training/after_optimisation' is HEALTHY
```

Figure.8 Health status of the result



```
ting after reserving slots (ms)=0
17/04/27 01:06:43 INFO mapred.JobClient:     Total time spent by all maps waitin
g after reserving slots (ms)=0
17/04/27 01:06:43 INFO mapred.JobClient:     Launched map tasks=1
17/04/27 01:06:43 INFO mapred.JobClient:     Data-local map tasks=1
17/04/27 01:06:43 INFO mapred.JobClient:     SLOTS_MILLIS_REDUCES=10019
17/04/27 01:06:43 INFO mapred.JobClient:   FileSystemCounters
17/04/27 01:06:43 INFO mapred.JobClient:     FILE_BYTES_READ=501958
17/04/27 01:06:43 INFO mapred.JobClient:     HDFS_BYTES_READ=1977767
17/04/27 01:06:43 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=1113974
17/04/27 01:06:43 INFO mapred.JobClient:     HDFS_BYTES_WRITTEN=888
17/04/27 01:06:43 INFO mapred.JobClient:   Map-Reduce Framework
17/04/27 01:06:43 INFO mapred.JobClient:     Reduce input groups=36
17/04/27 01:06:43 INFO mapred.JobClient:     Combine output records=0
17/04/27 01:06:43 INFO mapred.JobClient:     Map input records=21215
17/04/27 01:06:43 INFO mapred.JobClient:     Reduce shuffle bytes=501958
17/04/27 01:06:43 INFO mapred.JobClient:     Reduce output records=36
17/04/27 01:06:43 INFO mapred.JobClient:     Spilled Records=42430
17/04/27 01:06:43 INFO mapred.JobClient:     Map output bytes=459522
17/04/27 01:06:43 INFO mapred.JobClient:     Combine input records=0
17/04/27 01:06:43 INFO mapred.JobClient:     Map output records=21215
17/04/27 01:06:43 INFO mapred.JobClient:     SPLIT_RAW_BYTES=102
17/04/27 01:06:43 INFO mapred.JobClient:     Reduce input records=21215
```

training@localhost:~/workspace
File Edit View Terminal Tabs Help

```
ting after reserving slots (ms)=0
17/04/30 23:46:18 INFO mapred.JobClient:     Total time spent by all maps waitin
g after reserving slots (ms)=0
17/04/30 23:46:18 INFO mapred.JobClient:     Launched map tasks=1
17/04/30 23:46:18 INFO mapred.JobClient:     Data-local map tasks=1
17/04/30 23:46:18 INFO mapred.JobClient:     SLOTS_MILLIS_REDUCES=8652
17/04/30 23:46:18 INFO mapred.JobClient:   FileSystemCounters
17/04/30 23:46:18 INFO mapred.JobClient:     FILE_BYTES_READ=501958
17/04/30 23:46:18 INFO mapred.JobClient:     HDFS_BYTES_READ=1977774
17/04/30 23:46:18 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=1114010
17/04/30 23:46:18 INFO mapred.JobClient:     HDFS_BYTES_WRITTEN=888
17/04/30 23:46:18 INFO mapred.JobClient:   Map-Reduce Framework
17/04/30 23:46:18 INFO mapred.JobClient:     Reduce input groups=36
17/04/30 23:46:18 INFO mapred.JobClient:     Combine output records=0
17/04/30 23:46:18 INFO mapred.JobClient:     Map input records=21215
17/04/30 23:46:18 INFO mapred.JobClient:     Reduce shuffle bytes=501958
17/04/30 23:46:18 INFO mapred.JobClient:     Reduce output records=36
17/04/30 23:46:18 INFO mapred.JobClient:     Spilled Records=42430
17/04/30 23:46:18 INFO mapred.JobClient:     Map output bytes=459522
17/04/30 23:46:18 INFO mapred.JobClient:     Combine input records=0
17/04/30 23:46:18 INFO mapred.JobClient:     Map output records=21215
17/04/30 23:46:18 INFO mapred.JobClient:     SPLIT_RAW_BYTES=109
17/04/30 23:46:18 INFO mapred.JobClient:     Reduce input records=21215
```

Figure.9 Performance - before and after optimization

SLOTS MILLIS REDUCES is the total time spent by all reduces in the occupied slots. Difference in the time taken for running reduces tasks before and after optimisation is noted to be 1367ms as shown in Figure 9. After optimization it takes less time.

6.2 Performance Analysis

The performance of the job for the same input is analysed before and after the optimisation is show in Table 1. Job duration, Map task time and Reduce task time are taken as the factors for the performance analysis. The time taken to complete the job is less after optimization is shown in Figure 10. The performance has evidently improved after the optimisation.

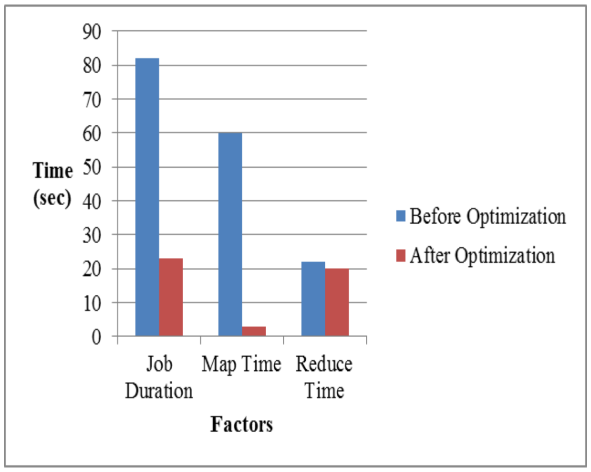


Figure.10 Performance Evaluation

7. Conclusion

In this article, an improved hadoop framework is presented. It includes analysing the internal working of the Hadoop MapReduce job, so that changes are done to the job in progress, task in progress and task tracker files. In H2Hadoop [19] the work was concentrated on the DNA dataset only and in the improved hadoop [22] they have considered the text data and the performance time was not reasonably improved. The changes performed to optimise the system in this article are in contact with the job tracker at once, launching synchronised cleanup function, cleanup later checkup, skipping feature, resetting the heartbeat interval from the response, avoiding speculative tasks resulted in an improved hadoop framework. The performance of the improved system using optimization is better in the computation time.

8. Future Work

Performance factors

The other performance factors of File System Counters, MapReduce Framework can be modified to obtain a more optimised hadoop system.

Tool for better performance

The result can be analysed using Spark to obtain a faster performance than Hadoop, as spark runs in-memory on the cluster, and it is not tied to Hadoop MapReduce paradigm. This makes repeated access to the same data much faster. Spark can run as a standalone or on top of Hadoop YARN, where it can read data directly from HDFS.

Security issues in Hadoop can be addressed at various levels, including but not limited to file system, networks, scheduling, load balancing, concurrency control, and databases.

References

[1] MMing, M., G. Jing, and C. Jun-jie, "Blast-Parallel: The parallelizing implementation of sequence alignment algorithms based on Hadoop platform", *In: Proc. of International Conf. on Biomedical Engineering and Informatics (BMEI)*, IEEE, pp. 465-470, 2013. doi**:** 10.1109/BMEI.2013.6746988

[2] cC. Schatz, B. Langmead and S. L. Salzberg, "Cloud computing and the DNA data race", *Journal of Nature biotechnology*, Vol.*28, No.*7, p.691, 2010. doi:10.1038/nbt0710-691.

[3] E. E. Schadt, et al., "Computational solutions to large-scale data management and analysis", *Journal of Nature Reviews Genetics*, Vol.11, No.9, pp. 647-657, 2010. https://doi.org/10.1038/nrg2857.

[4] Marx, *Biology: The big challenges of big data*. Nature Publishing, 498(7453): pp. 255-260, 2013. https://doi.org/10.1038/498255a

[5] N. Tiwari, S. Sarkar, U. Bellur and M. Indrawan, "Classification framework of MapReduce scheduling algorithms", *International Journal of ACM Computing Surveys (CSUR)*, Vol.47, No.3, p.49, 2015.doi:10.1145/2693315

[6] M. Zaharia, et al., *Job scheduling for multi- user mapreduce clusters*, Tech. Rep. UCB/EECS-2009-55, EECS Department, University of California, Berkeley, 2009. doi =10.1.1.649.3097

[7] R. Gu, et al., "SHadoop: Improving MapReduce performance by optimizing job execution mechanism in Hadoop clusters", *Journal of Parallel and Distributed Computing*, Vol.74, No.3, pp. 2166-2179, 2014. https://doi.org/10/1016/j.jpdc.2013.10.003

[8] Q. Chen, C. Liu, and Z. Xiao, "Improving MapReduce performance using smart speculative execution strategy", *Journal of IEEE Transactions on Computers*, Vol.*63, No.*4, pp.954-967, 2014. doi:10.1109/TC.2013.15

[9] Y. Jinshuang, et al., "Performance Optimization for Short MapReduce Job Execution in Hadoop", In: *Proc. of International Conf. on Cloud and Green Computing (CGC)*, IEEE, pp. 688-694, 2012. doi:10.1109/CGC.2012.40

[10] Apache, Centralized Cache Management in HDFS. Update date 2014.https://hadoop.apache.org

[11] S. Zhang, et al., "Accelerating MapReduce with distributed memory cache", In: *Proc. of International Conf. On Parallel and Distributed Systems (ICPADS)*, IEEE, pp.472-478, 2009. doi:10.1109/ICPADS.2009.88

[12] J. Zhang, G. Wu, X. Hu and X. Wu, "A Distributed Cache for Hadoop Distributed File System in Real-Time Cloud Services", *In: Proc. of International Conf. On Grid Computing,* IEEE Computer Society, pp.12-21, 2012. doi:10.1109/Grid.2012.17

[13] L. Longbin, et al., "ShmStreaming: A Shared Memory Approach for Improving Hadoop Streaming Performance", In: *Proc. of International Conf. On* Advanced Information Networking and Applications (AINA), IEEE, pp.137-144, 2013. doi:10.1109/AINA.2013.90

[14] B. Palanisamy, et al., "Purlieus: locality-aware resource allocation for MapReduce in a cloud.", *In: Proc. of International Conf. On High Performance Computing, Networking, Storage and Analysis*, ACM, pp.1-11, 2011. doi:10.1145/2063384.2063462

[15] H. Alshammari, J. Lee and H. Bajwa, "H2Hadoop: Improving Hadoop Performance Using the Metadata of Related Jobs," in *IEEE Transactions on Cloud Computing*, vol. 6, no. 4, pp. 1031-1040, 2016.doi:10.1109/TCC.2016.2535261

[16] Hammoud, and M. F. Sakr, "Locality-aware reduce task scheduling for MapReduce", *In: International Conf. on Cloud Computing Technology and Science (CloudCom),* IEEE, pp. 570-576, 2011. doi:10.1109/CloudCom.2011.87

[17] J. B. Buck, et al., "SciHadoop: Array-based query processing in Hadoop", *In: Proc. of International Conf. on High Performance Computing, Networking, Storage and Analysis (SC)*, 2011. doi:10.1145/2063384.2063473

[18] Holmes, *Hadoop in Practice*, Manning Publications Co., 2012.

[19] Edward, W. Dean and R. Jason, *Programming hive*, O'Reilly Publications, 2012.

[20] Y. Xiao and B. Hong, “Bi-Hadoop: Extending Hadoop to Improve Support for Binary-Input Applications”, In: *Proc. of International Symposium On* Cluster, Cloud and Grid Computing (CCGrid), IEEE/ACM, pp.245-252, 2013. doi:10.1109/CCGrid.2013.56

[21] https://hadoop.apache.org/docs/r2.7.1/api/org/apache/hadoop/mapreduce/JobCounter.html

[22] Thanekar, Sachin & Bagwan, A. & Subrahmanyam, K..,Improving Hadoop performance by enhancing name node capabilities.Fronteiras. 6. 1-18.,2016. doi:10.21664/2238-8869.