International Journal of Applied Information Systems (IJAIS) – ISSN : 2249-0868
Foundation of Computer Science FCS, New York, USA
Volume 4– No.7, December 2012 – www.ijais.org

Performance Overhead on Relational Join in Hadoop using Hive/Pig/Streaming - A Comparative Analysis

Prabin R. Sahoo
Tata Consultancy Services
Yantra Park, Thane
Maharashtra, India

ABSTRACT

Hadoop Distributed File System (HDFS) is quite popular in the big data world. It not only provides a framework for storing data in a distributed environment, but also has set of tools to retrieve and process these data using map-reduce concept. This paper discusses the result of evaluation of major tools such as Hive, Pigand hadoop streaming for solving problems from a relational prospective and comparing their performances. Though big data cannot be compared to the strength of relational database in solving relational problems, but as big data is about data so the relational nature of data access cannot be eliminated altogether. Fortunately, there are ways to deal with this which has been discussed in this paper from a performance prospective. This may help the big data community in understanding the performance challenges so that further optimization can be done and the application developers' community can learn how strategically the relational operations need to be used.

General Terms

Hive, Pig, Hadoop, HDFS, Map-Reduce, streaming.

1. INTRODUCTION

Big data plays a key role in processing data with varied structures. This is unlike the way the relational mapping happens in traditional databases, where data structure is predefined. However, though big data has its own merits and demerits, and its strength lies in working on handling huge volume of data but it is worthwhile to understand how big data solves problems with varieties of data.

Essentially big data works on a distributed environment where the data storage is spread across several computing nodes running in clusters. This is due to the fact that huge volume of files containing data can be stored in the node in a distributed manner. Since big data involves files, so big data is about processing files in a distributed environment. For example, if there is a file which contains the web access logs from across the globe, it is possible to determine which region has maximum access to a given web site.

Though big data covers a wide range of underlying data storage and map-reduce framework but the focus in this paperis on Hadoop map-reduce framework [1], storage as HDFS and tools such asPig, Hive and Hadoop streaming. A relational problemhas been discussed along with the experiments to demonstrate various approaches to solve it using the above available framework and tools.

1.1 Hadoop Distributed File System

Hadoop Distributed File System [7, 10]comprises of name node, data nodes, and secondary name node. The name node contains the metadata information about all the files, directories, permissions, storage details and which data nodes contain those. For example, a file can be copied from local directory /home/user/web/server.log to a HDFS directory as follows.

bin/hadoop dfs –copyFromLocal /home/user/web/server.log hdfs://localhost:9000/weblogDir

This command copies is the server.log file in HDFS into a given directory *weblogDir*.

If the *weblogDir* does not exist, then this need to be created first,otherwise it is overwritten with the file 'server.log'. The directory can be created using the following command from the hadoop home directory.

bin/hadoop dfs –mkdir hdfs://localhost:9000/weblogDir

When the directory is created the name node keeps the metadata such as directory path in HDFS, its permissions, and when a file is copied to this directory, the name node provides other information such as the data nodes where the file needs to be stored and their replications in the neighborhood nodes.

1.2 Map-Reduce

Map-Reduce [11]is the computation framework which works on the distributed data. It comprises of a job tracker and several task tracker. The job tracker runs as a centralized process, whereas task tracker run in each data node. The job tracker creates tasks and assigns these to task tracker for computation. The task tracker runs the assigned tasks locally. Once the map task is completed the output is collected in the reduce operation. This operation is known as map-reduce operation.

1.3 Relational Operations

Relational operations are set of operators that act on a domain of relations. In this paper the experiment is being done to demonstrate relational join operation joining two files on a common key.

2. LITERATURE REVIEW

Relational solutions with join operator in big data have been discussed earlier in a number of papers such as skew algorithm [2]. However, other relational join cases such as Hive [8],Pig [9] and Hadoop streaming needs to be discussed as well as these are popular tools of hadoop. Therefore it is important to have clarities on available solutions which can be

easily maintainable, usable. In this paper the interest is more on the findings of available frameworks and their performance overheads to solve relational problems. In this connection, few papers were searched for finding such comparisons. However, the comparison of performance with Hive, Pig, streaming along with its simplicities was not found. Since Hive and Pig are easily available, for the developers' community it is important to see how Hive and Pig perform along with relational join. In [3] the Facebook Data Infrastructure team has published the relational concept in Hive. They have demonstrated how to create relational tables in Hive and how map-reduce works on it along with the Query compilation and optimization. Their paper is interesting for understanding the design concept of Hive using hadoop and it gives sufficient directions to apply this on the problem domain discussed in this paper. In [4] the author has mentioned about hadoop configuration parameters such as setting of parallel mapper tasks, parallel reducer tasks which can help optimizing the processing time. In [5] the author has mentioned about Hadoop framework at beginners level which helps in understanding the building blocks of hadoop architecture. In [6] authors have mentioned about the architecture of Hadoop Distributed File System. This is important as HDFS is the underlying storage framework for Hive, Pig and Hadoop streaming for retrieving the data for relational join operation.

3. CASE STUDY

The case study has been under taken for the purpose of demonstrating how to solve relation problems. The data used in the experiment is fabricated. Therefore, the inference out of data analysis does not represent real consequence. But it gives a close impression how various data analysis can be done using Pig, Hive, Hadoop streaming and the performance metrics of each framework provides guidance to choose the appropriate framework.

3.1 Scenario

A fund management company has various schemes on which the investors have invested on schemes. The investor needs to register on its site before he invests in any scheme. During the registration process the investor needs to enter his details such as name, address, interest, profession, age etc. Once the registration process is completed the investor can invest into one or more schemes. During the investment phase, the scheme-id, investment amount is captured for the investor.

Input:

There are 2 log files which captures these logs.

A) Personal details

B) Investment details.

These two logs files are copied to HDFS.

Output:

Find out the average investment per income group. Other information such as to find out the age group that invests more on a given scheme and so on.

The file looks as follows:

Personal Detail log contains

Name, Memberid, age, address, profession, income group, interest

XYZ, **1234,25,5** Dela Road, engineer, 10000-20000, trading

Investment log contains

memberid, transactionid, schemeid, amount,date,unit price

1234, 8675, 101, 10000, 20120922,500

Solution:

1. In order to find the average investment per income group
    a. join the two log files (as only registration does not guarantee that the investor has invested)
    b. group by income group

The complex part in this solution is the joining of two files. So the focus is limited to the join functionality.

Table 1. Experimental setup

Files	Personal.log, investment.log
Number of data nodes	3
Number of cores	24
CPU configuration of each DataNode, NameNode	2.79 GHz, AMD Opteron 8439 SE
Memory of each data node/namenode	8 GB
Number of records in personal.log	76,800,000
Number of records in investor.log	76,800,000
meberid distribution	random
schemeid distribution	random
HDFS chunk size	64MB
HDFS version	0.1.3
Hive version/Pig	0.9.0/0.10.0

3.2 Hadoop streaming approach

In hadoop approach hadoop streaming has been used to join the two files, and after that a java program has been developed using map-reduce interface to compute the average on the investment per income group. In a key-value context, the key in this experiment is the income group and the value is the investment amount. The stream input and parameters are given as follows.

```
-input hdfs://localhost:9000/myinput/personal.log \
-input hdfs://localhost:9000/myinput/investment.log \

 -output output20 \
-jobconf stream.map.output.field.separator=, \
```

```
-jobconf stream.num.map.output.key.fields=1 \
-jobconf mapreduce.map.output.key.field.separator=, \
-jobconf mapreduce.partition.keypartitioner.options=-k1 \
-jobconf mapreduce.job.reduces=8 \
-mapper map.pl \
-reducer reduce.pl \
-file map.pl \
-file reduce.pl \

-practitioner
org.apache.hadoop.mapred.lib.KeyFieldBasedPartitioner
```

3.2.1 Mapper: map.pl

The mapper script is simple. Since the join has to be on the common key, the memberid has been chosen from both the files. Since the memberid is available on the second field of the personal log file the mapper is doing a swap on the memberid when it finds the records from the personal log. The pseudo code is as follows.

```
while(<STDIN>){
chomp;
my@temp = split",",$_;
if(scalar(@temp) == 6){

## Record from investment log, just print it
  print"$_\n";
}else{

## Record from personal.log swap the firstand 2nd field.
      my$temp = $temp[0];
      $temp[0] = $temp[1];
      $temp[1] = $temp;
            my$str = join(",",@temp);
      print"$str\n";
 } }
```

3.2.2 Reducer: reduce.pl

The following logic joins the records in the given key. Since the input involves two files, the logic takes care of the records in which order those appear. The pseudo code is as follows.

```
my$prevKey;
my $prevRow1;

my $prevLength;
while(<STDIN>){
chomp;
if(/^(\d+)\s+(.*)/){
   my $key = $1;
   my $value = $2;
   if(! defined $prevKey || ! defined $prevRow) {

    ## First time these need to be initialized
    $prevKey = $key;
    $prevRow = "$key, value";

my @temp1 = split ",", $prevRow;

    $prevLength = $#temp1+1;

    next;
   }else{
     if($prevKey == $key){

          my @temp = split ",",$value;

     ## do not join multiple records of same memberid
     fromsame file.

      next if $#temp+1 == $prevLength;


       if(($#temp+1)> 6){

## Record is from personal.log, so need to keep the fields
before the second record from investment log
         print "$key,$value,$prevRow\n";
       }else{
         print "$prevRow,$key,$value\n";
       }
       next;
     }
   }
          $prevKey = $key;
          $prevRow = "$key,$value";

my @temp1 = split ",",$prevRow;

     $prevLength = $#temp1+1;


     }
    }
```

The join is completed; records from two files have been joined. The average investment per income group has been found using the map and reduce library. The pseudo code snippet is as follows.

```
      public void map (LongWritable key, Text value,
OutputCollector<Text, DoubleWritable> output, Reporter
reporter) throws IOException {

     String line = value.toString();

     String [] temp = line.split(",");

     double x = Double.parseDouble(temp[4]);

     mytext.set(temp[6]);

     one.set(x);

     output.collect(mytext, one);

    }

   public void reduce(Text key, Iterator<DoubleWritable>
values, OutputCollector<Text, DoubleWritable> output,
Reporter reporter) throws IOException {

     double sum = 0.0;

    int count = 0;

     while (values.hasNext()) {

      sum += (double) values.next().get();

      count++;

     }

    double avg = sum/count;

     output.collect(key, new DoubleWritable(avg));

    }
```

3.2.3 Pig approach

In Pig approach [9], we can load the file as per the following pseudo code. This is purely for explanation purpose.

A = LOAD '/personal.log' USING PigStorage(',') AS (name:chararry,memberid:long,….,…,….,…… incomegroup:chaarray,…);

B = LOAD '/investor.log' USING PigStorage(',') AS (memberid:long,….,…,investment:double,…);

C = JOIN A BY $1, B BY $0;

D =GROUP C BY incomegroup;

E = FOREACH DGENERATEuser, AVG (C.investment);
F = ORDER E BY $0;
store F INTO '/tmp/income';

With this approach each time a new query is required, first the join will happen and after that the processing. So to avoid that it is better to have only join through Pig once and later on it Hadoop's map and reduce java interface can be used or even Python or PERL in the hadoop streaming. In the experiment the following Pig statements are used.

A = LOAD '/personal.log' USING PigStorage(',') AS (name:chararry,memberid:long,….,…,….,

incomegroup:chaarray,…);

B = LOAD '/investor.log' USING PigStorage(',') AS (memberid:long,….,…,investment:double,…);

C = JOIN A BY $1, B BY $0;

STORE C USING PigStorage(',') INTO 'outputDir/personal.investor.log'

This needs to be run just once. This would create a single file joined on the memberid.

3.2.4 Hive approach

Hive [8] helps programmers/data scientists to write queries in a simple way. In the example, a Hive query can be written as below. The query is based on the inner join.

Hive> CREATE TABLE PERSONALDETAILS (NAME STRING, MEMBER BIGINT,…,……,STRING INCOMEGROUP…..) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';

Hive> LOAD DATA INPATH '/PERSONAL.LOG' OVERWRITE INTO TABLE PERSONALDETAILS;

Hive> CREATE TABLE INVESTORDEATILS (MEMBER BIGINT, TRANSACTIONID BIGINT,…, AMOUNT DOUBLE…..) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';

Hive> LOAD DATA INPATH '/INVESTMENT.LOG' OVERWRITE INTO TABLE PERSONALDETAILS;

Using the Hive JDBC and java program the following query can be executed.

String sql = "SELECT PERSONALDETAILS.incomegroup, AVG(INVESTORDEATILS.amount)) FROM PERSONALDETAILS JOIN INVETSTORDETAILS ON (PERSONALDETAILS.member=INVESTORDETAILS.me mberid) GROUP BY PERSONALDETAILS.incomegroup;";

ResultSet res = stmt.executeQuery(sql);

while (res.next()) { // extract and write it to a file }

4. RESULTS

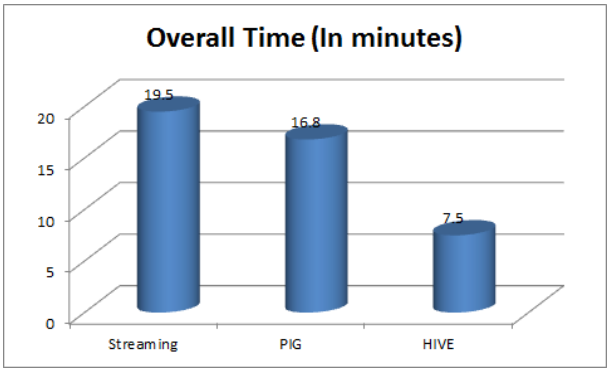

Fig. 1 Time taken to join 2 files on a given key with records 76,800,000 records in each file. Records in each file are in random order.

The execution times are listed in Fig 1. Hadoop streaming is taking 19.5 minutes, Pig is taking 16.8 minutes to join. Hive is taking 7.5 minutes to complete the query. Fig 2 shows memory utilizations by Hive which is comparatively low than utilization by hadoop streaming and Pig.The detailed memory usages by each are given in fig 3, fig 4 and fig 5.

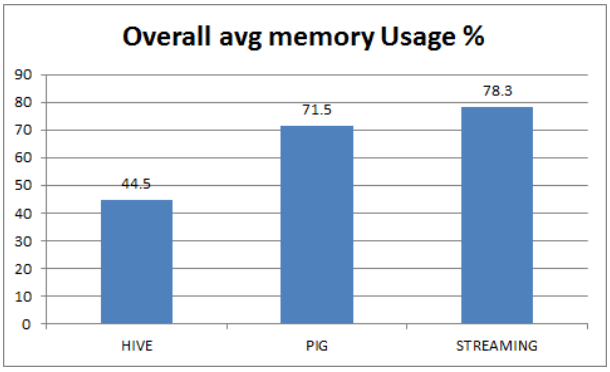

Fig. 2 Overall average memory utilizations using Hive, Pig, Hadoop Streaming for join of 2 files on a given key with records 76,800,000 records in each file. Records in each file are in random order

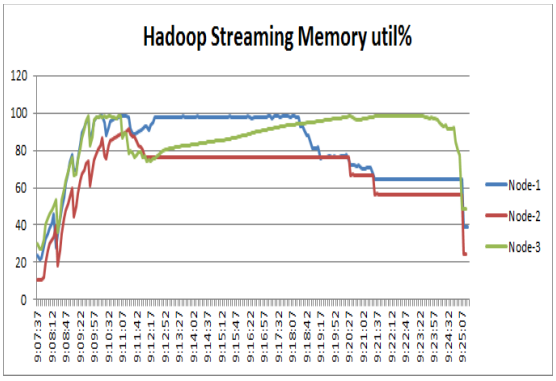

Fig. 3 Memory utilization in all of the nodes using Hadoop Streaming for join of 2 files on a given key with records 76,800,000 records in each file. Records in each file are in random order

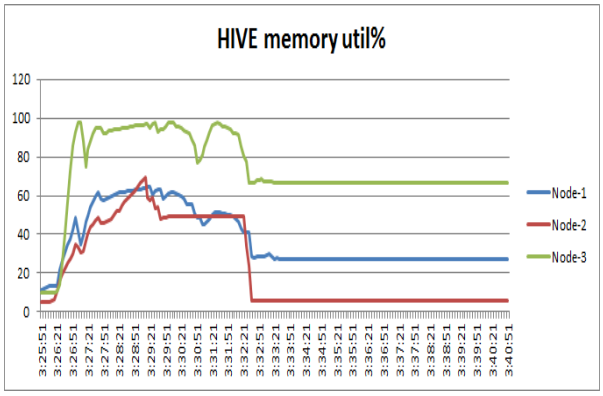

Fig. 4 Memory utilization in all of the nodes using Hivequery for join of 2 files on a given key and processing with records 76,800,000 records in each file. Records in each file are in random order

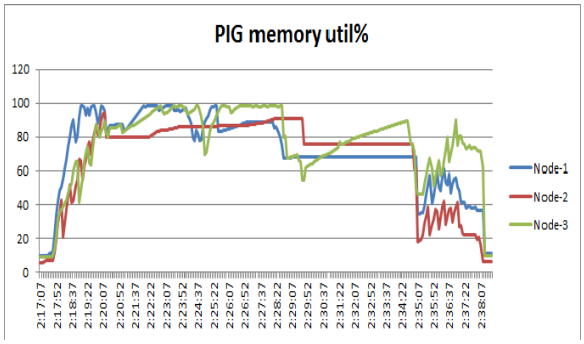

Fig. 5 Memory utilization in all of the nodes using Pig for join of 2 files on a given key with records 76,800,000 records in each file. Records in each file are in random order

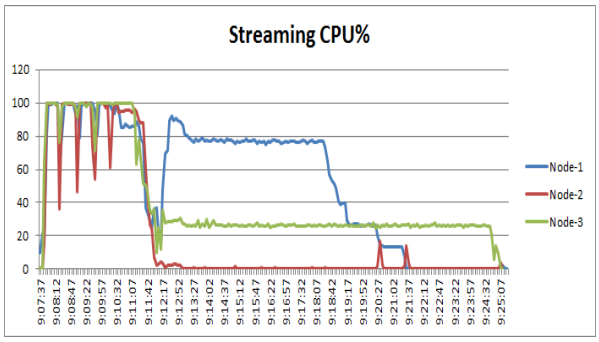

Fig. 6 CPU utilization in all of the nodes using streaming for join of 2 files on a given key with records 76,800,000 records in each file. Records in each file are in random order

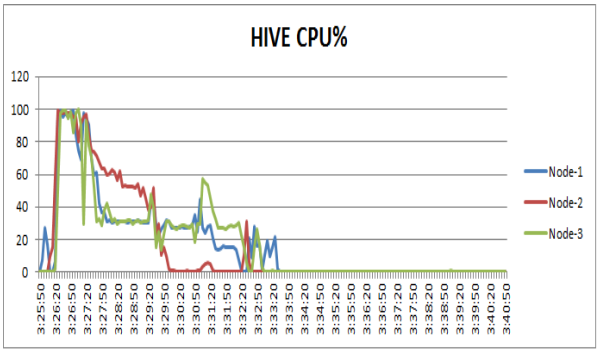

Fig. 7 CPU utilization in all of the nodes usingHive for join of 2 files on a given keywith records 76,800,000 records in each file. Records in each file are in random order

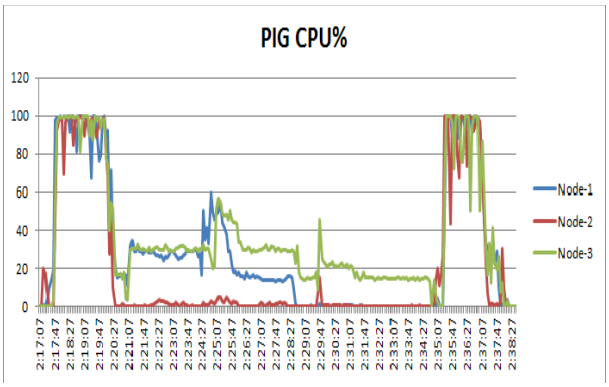

Fig. 8 CPU utilization in all of the nodes using Pig for join of 2 files on a given key with records 76,800,000 records in each file. Records in each file are in random order

5. CONCLUSION AND FUTURE WORK

Fig 1 shows relational JOIN is expensive using HADOOP framework such as Pig and streaming. Hivequery shows better result than streaming and Pig.Hive has taken just 7.5 minutes to join and process 76,800,000 records whereas streaming and Pig has taken 19.5 minutes and 16.8 minutes respectively for relational join. Pig and streaming shows high memory utilizations, compared to Hive as shown in fig 2. This indicates that during relational join, streaming and Pig needs high memory. The CPU utilizations in streaming and Pig show high utilizations as well. However, Pig involves simple steps compared to the streaming and Hive approach. Pig and streaming approach will be beneficial compared to Hive if the join is one time.In such case once the join is completed, subsequent queries can be fired on the combined files. Hiveapproach is beneficial compared to Pig and streaming if the one time join produces large number of fields in a record. In such case at a time fewer set of fields can be selected using Hive. Between Pig and streaming, streaming can provide more flexibility from programming prospective as PERL and PYTHON are widely used languages and provides rich regular expressions. Future work can be done on improvising Pig and streaming join algorithms and also other optimization techniques such as RCFile, partitioning, bucket can be evaluated for Hive.

6. REFERENCES

[1] Lucene Hadoop, "Hadoop Map-Reduce Tutorial", http://hadoop.apache.org/docs/r0.15.2/mapred_tutorial.html, retrieved online November 2012

[2] Viglas,S.D,Niazi,S, "SAND Join — A skew handling join algorithm for Google's Map/Reduce framework", http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=6151466&contentType=Conference+Publications&queryText%3Djoin+in+hadoop, Dec 2011

[3] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Ning Zhang, Suresh Antony, Hao Liu, and Raghotham Murthy, "Hive – A Petabyte Scale Data Warehouse Using Hadoop", infolab.stanford.edu/~ragho/Hive-icde2010.pdf,ICDE 2010

[4] Christer A. Hansen, "Optimizing Hadoop for the cluster", Institue for Computer Science, University of Troms0, Norway, http://oss.csie.fju.edu.tw/~tzu98/Optimizing%20Hadoop%20for%20the%20cluster.pdf, Retrieved online October 2012

[5] Nils Braden, "The Hadoop Framework", http://homepages.thm.de/~hg51/Veranstaltungen/MasterSeminar1011/NielsBraden.pdf, Retrieved September 2012

[6] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, Robert Chansler, The Hadoop Distributed File System, http://storageconference.org/2010/Papers/MSST/Shvachko.pdf, retrieved online October, 2012

[7] Apache Hadoop,"Hadoop 0.20 documentation", http://hadoop.apache.org/docs/r0.17.1/mapred_tutorial.html, August 2008, retrieved online August 2012

[8] Atlassian Confluence, "Hive Tutorial", https://cwiki.apache.org/Hive/tutorial.html, Feb 2011

[9] Apache Hadoop, "Pig 0.7.0 Documentation", http://Pig.apache.org/docs/r0.7.0/tutorial.html, retrieved online August, 2012

[10] HDFS Architecture Guide, http://hadoop.apache.org/docs/hdfs/current/hdfs_design.html, retrieved online October, 2012

[11] Map/Reduce Tutorial, http://hadoop.apache.org/docs/r0.20.2/mapred_tutorial.html, retrieved online October, 2012