Middle-East Journal of Scientific Research 24 (11): 3510-3514, 2016
ISSN 1990-9233

DOI: 10.5829/idosi.mejsr.2016.3510.3514

A New Proposed Hadoop Search (Hasear) Algorithm on Hadoop Framework Using Data Warehouse

T.K. Thivakaran and C. Jaya Kumar

Department of Computer Science and Engineering,
Sri Venkateswara College of Engineering, India

Abstract: The Big data Hadoop vendors are planning to embed with their products with Hadoop. The Hadoop framework has many tools along with the complete kit. So users can do any kind of workload by using these lists of packages. In many respects, the users are having lot of advantage to implement in their organizations with Business Intelligence (BI) and Data Warehousing (DW) concepts. The Hadoop vendors can integrate their user defined tools depends application requirements, hence it solves theanalytical functionality it currently lacks. In Apache foundation having more focus to satisfy their customers. The searching of data in Hadoop Distributed File System (HDFS) is not yet integrated with Hadoop framework, hence in this paper, proposed a new Hadoop Search (Hasear) tool with the existing Hadoop framework. This searching algorithm can give good performance with the traditional search algorithms, it can also search with various file formats and even current mailing services data. Hence in this paper, the proposal is to about new searching tool to search the data in the HDFS in various virtual or real machines. Hasear tool is integrated with the latest stable version of Hadoop 2.7.2.

Key words: Big data · Hadoop · Apache software · Business intelligence · Data warehouse

INTRODUCTION

The Big data is the upcoming filed of handling big data when not able to predict the decision with limited resources. Hadoop vendor Apache software foundation has various tools such as Hive, a higher level data access language [1], Hbase, Sqoop, etc. But the proposed tool must have minimum configuration with one megabyte to twentymegabyte of documents stored in Hadoop. The text documents in multiple formats such as email, doc, pdf, odt and metadata about those documents stored in SQL dB with sender, recipients, date and department etc., also the main source of documents may be emails and attachments. Now to the search the word or string in these documents, it needs to be able to do complex full text searches over those documents. Therefore to search the string in distributed file system using Hadoop framework is demonstrated. The Hadoop framework written in java allows distributed processing of large datasets across clusters of computers. In this case demonstrated using virtual machine based Hadoop cluster setup. The following sections will deal about how the distributed file system in large in size called big data can able to find a string with our proposed Hasear algorithm, which is embedded in Hadoop framework using data stored in Hadoop files system called data warehouse.

Related Work: The Apache package Foundation has multiple comes to handle Hadoop shortcomings within the arena of reportage and analytics.One promising project is Map Reduce is a parallel programming model for writing distributed applications devised at Google for efficient processing of large amounts of data, on large clustersof commodity hardware in a reliable, fault tolerant manner. The Map Reduce program runs on Hadoop which is an Apache open-source framework.

Hive gives SQL-like access to Hadoop, however it will nothing to beat its guideline execution worldview. Another is HBase, that beats Hadoop's dormancy issues however is implied for snappy line based peruses and

Corresponding Author: T.K. Thivakaran, Department of Computer Science and Engineering,
Sri Venkateswara College of Engineering, India.

keeps in touch with bolster predominant value-based applications. every produce table-like structures on prime of Hadoop records [2]. Another abnormal state scripting dialect is Pig Latin that like Hive keeps running on Map Reduce, however is most appropriate to assembling progressed, parallelized data streams, in this way its one thing to that ETL software engineers might float [3].

Another rising Apache extend that addresses the deficiency of sacred data in Hadoop is HCatalog that takes Hive's data and makes it a ton of by and large reachable over the Hadoop plan, together with Pig, Map Reduce and HBase. A little while later SQL sellers will be ready to address HCatalog with Map Reduce, net and Java data property (JDBC) interfaces to see the structure of Hadoop data before dispatching an inquiry. Equipped with this data, SQL stock will be ready to issue united inquiries against SQL and Hadoop data sources and return a brought together result set while not clients grasping wherever data lies or the best approach to get to it. On the information mining front, Apache driver could be an arrangement of information digging calculations for agglomeration; grouping and cluster construct helpful sifting that keep running with respect to Map Reduce.

At long last, Apache Sqoop could be a venture to quicken mass masses in the middle of Hadoop and relative information bases, information body procedure and Apache Flume streams monstrous volumes of log data from different sources into Hadoop [4].

These comes are still inside of the early stages, however a few driving edge adopters have implemented them and are giving abundant criticism, if not code, to the Apache group to make these code situations endeavour prepared. they're force helped by a few modern open supply merchants that contribute the greater part of the code to Apache Hadoop comes and need to make a chic, endeavour prepared plan [5].

The term “text search” is not new to implement in any of the file systems. The searching of data with small file system and big file system has different time complexity and memory complexity. The time taken to search the string having classic problem with matching the single and multiple terms of the same and count is needed. Therefore searching the text in Hadoop distributed file system is not yet introduced any of the tools till now, instead this can be handled using grep command in terminal mode, even having limitations of file size and connection issues. Hence this paper proposed a new Hasear algorithm embedded in Hadoop framework.

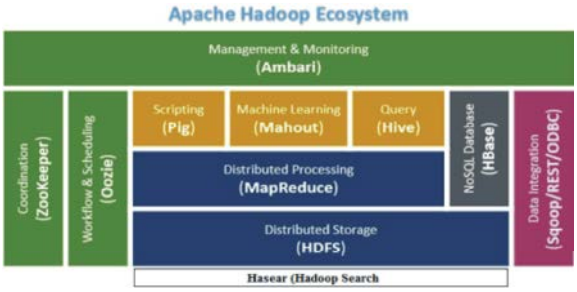

Fig. 1: Hadoop Framework with Haser Tool Integration

Hasear Algorithm in Hadoop Framwork: A real world application for this situation is matching variety of medical treatments against an inventory of medical conditions and searching for that treatments discuss that conditions. Another example is traversing an oversized assortment of judicial precedents and extracting the laws they reference [6].

The most fundamental methodology is to circle through the search queries and hunt through the content every expression, one by one. This methodology does not scale well. Hunting down a string inside another has the many-sided quality O(n). Rehashing that for "m" search queries prompts the terrible O(m*n). The most ideal arrangement would be one that navigates the content body just once. This requires search queries to be listed in a structure that can be transverse directly, in parallel with the content body, in one pass, accomplishing a last intricacy of O(n).

The benefit of a Hasear is that it altogether cuts look time. To make it less demanding to get a handle on for the reasons of this Hasear, navigating a double tree has the unpredictability of O(log2n), since every hub branches into two, slicing the remaining traversal down the middle. All things considered, a ternary tree has the traversal many-sided quality of O(log3n) [7]. In a Hasear, in any case, the quantity of youngster hubs is managed by the succession it is speaking to and on account of clear/important content, the quantity of kids is typically high.

Existing Hadoop Framework Projects Overview: A hefty portion of the associations as of now utilizing Hadoop as a part of their current BI situations, it's essential to comprehend the extent to which BI and DW items can interoperate with Hadoop. There are four sorts of reconciliation, they are Connectivity, Hybrid frameworks, Native Hadoop and Interoperability [8]. The Connectivity

is to Prebuilt information connectors empower engineers to view and move information in mass between the two situations. The Hybrid frameworks used to mix SQL and Map Reduce usefulness into a solitary information preparing environment. The Native Hadoop used to run locally on Hadoop however are gotten to by means of a graphical client interface that conceals the complexities of the basic preparing environment. The interoperability is an application programming interfaces and metadata lists empower engineers in one environment to outline and powerfully execute local employments in another environment.

Proposed Hasear tool Integrated on Hadoop Framework: The searching of the string in a small file has less time with system configuration and it various depends upon the big data applications. So the big data has more focus on data analytics, the Hadoop is the framework can handle any type of analysis against data. These applications are more powerful than more useful to predict the data with good decision making. After the introduction of distributed file systems more data can be handled efficiently, located in various regions of the world. So Hadoop can give better platform to the end user to handle these situations. So in this paper we discussed the need of searching a string in distributed file system. But in this proposal the data string is searching in virtual machine based distributed file system with limited resources. Hence this can be extended in future to implement the same in real time machines. The following program used to embed in the Hadoop framework with the version 2.7.2. The Hasear program and the package is given below and the searching is demonstrated in Figure 2.

Hasear Program

```
"packagehadoop.hasear;"
"importjava.io.IOException;"
"importjava.util.StringTokenizer;"
"importorg.apache.hadoop.conf.Configuration;"
"importorg.apache.hadoop.fs.Path;"
"importorg.apache.hadoop.io.IntWritable;"
"importorg.apache.hadoop.io.Text;"
"importorg.apache.hadoop.mapreduce.Job;"
"importorg.apache.hadoop.mapreduce.Mapper;"
"importorg.apache.hadoop.mapreduce.Reducer;"
"importorg.apache.hadoop.mapreduce.lib.input.FileInput
tFormat;"
"importorg.apache.hadoop.mapreduce.lib.output."
FileOutputFormat;"
"public class HasearImpl" {
 "public static class TokenizerMapper
extends Mapper<Object, Text, Text, IntWritable>{
private Hasear hasear;
private static IntWritable offset;
Private Text offsetFound = new Text ("offset");"

"Public void map (Object key, Text value, Context context
 ) throws IOException, InterruptedException {
StringTokenizeritr = new
StringTokenizer(value.toString());"
"while (itr.hasMoreTokens()) {
 String line = itr.nextToken();
int offset1 = hasear.search(line);
if (line.length() != offset1) {
offset = new IntWritable(offset1);
context.write(offsetFound,offset);
 }} }"
"@Override
Public final void setup (Context context) {
if (hasear == null)
hasear = new
"Hasear(context.getConfiguration().get("pattern"));
 }}"
"public static void main(String[] args) throws Exception {
 Configuration conf = new Configuration();
conf.set("pattern","your_pattern_here");
 Job job = Job.getInstance(conf, "Hasear");
job.setJarByClass(HasearImpl.class);
job.setMapperClass(TokenizerMapper.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);
FileInputFormat.addInputPath(job, new Path(args[0]));
FileOutputFormat.setOutputPath(job, new Path(args[1]));
System.exit(job.waitForCompletion(true) ? 0 : 1);
 }}"
```

Hasear Package

```
"importjava.io.IOException;"
"importjava.util.*;"
"importorg.apache.hadoop.conf.*;"
"importorg.apache.hadoop.fs.*;"
"importorg.apache.hadoop.conf.*;"
"import org.apache.hadoop.io.*;"
"importorg.apache.hadoop.mapreduce.*;"
"importorg.apache.hadoop.mapreduce.lib.input.*;"
"importorg.apache.hadoop.mapreduce.lib.output.*;"
"importorg.apache.hadoop.util.*;"

"public class StringMatching extends Configured
```

```
implements Tool {“
“public static void main(String args[]) throws Exception {
long start = System.currentTimeMillis();
int res = ToolRunner.run(new StringMatching(), args);
long end = System.currentTimeMillis();
System.exit((int)(end-start));”
 }
“publicint run(String[] args) throws Exception {
 Path inputPath = new Path(args[0]);
 Path outputPath = new Path(args[1]);
 Configuration conf = getConf();”
“Job job = new Job(conf, this.getClass().toString());
FileInputFormat.setInputPaths(job, inputPath);
FileOutputFormat.setOutputPath(job, outputPath);
job.setJobName("StringMatching");
job.setJarByClass(StringMatching.class);
job.setInputFormatClass(TextInputFormat.class);
job.setOutputFormatClass(TextOutputFormat.class);
job.setMapOutputKeyClass(Text.class);”
“job.setMapOutputValueClass(IntWritable.class);”
“job.setOutputKeyClass(Text.class);”
“job.setOutputValueClass(IntWritable.class);”
“job.setMapperClass(Map.class);”
‘job.setCombinerClass(Reduce.class);”
“job.setReducerClass(Reduce.");”
 “returnjob.waitForCompletion(true) ? 0 : 1;”
 }
“public static class Map extends Mapper<LongWritable, Text, Text, IntWritable>” {
“private final static IntWritable one = new IntWritable(1);
private Text word = new Text();”
“@Override
public void map(LongWritable key, Text value, Mapper.Context context) throws IOException, InterruptedException” {
“String line = value.toString();
StringTokenizertokenizer = new StringTokenizer(line);
while (tokenizer.hasMoreTokens()) {
word.set(tokenizer.nextToken());
context.write(word, one);”
 }} }
“public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable>” {
 @Override
“public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {
 Hasear bm = new Hasear(); “
“ b o o l e a n  f l a g  = bm.findPattern(key.toString().trim().toLowerCase(), "abc");
if(flag){
context.write(key, new IntWritable(1));
}else{
context.write(key, new IntWritable(0));
 } } } }”
```

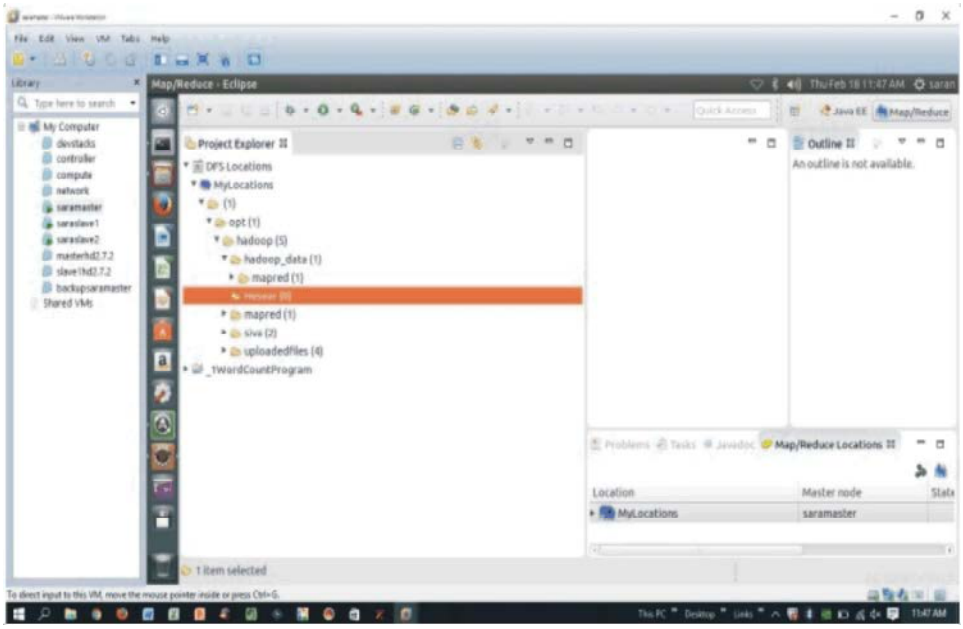

Fig. 2: Hadoop Framework with Heaser Tool Integration

Ordinarily the Hadoop information put away in neighbourhood framework can be downloaded utilizing program settings and the remote information need to check the name node association with the data node in framework. Following Hadoop stores information in a dispersed record framework, the data association can happens with group of documents with Hadoop's name node. Thus the seeking of string with one data node with another is straightforward, if the group of nodes more often than not executes Hadoop recording framework charges to recover, for example, documents circulated over the bunch and move them to the asking for framework.

The benefit of information connectors is that they're similarly clear to make and utilize. The drawbacks territory unit that they require moving whole learning sets, that in extremely huge information setting isn't perfect. On the off chance that the information volumes region unit sizeable, the exchange may take an extended time and make system bottlenecks. chiefs UN office wish to move information in the middle of Hadoop and PC database administration frameworks can utilize Sqoop or Flume or some exclusive mass burden utility rather than seek the string abuse these instruments can enhance the execution.

CONCLUSION

The Hadoop framework with Hasear tool will give benefit to large databases stored in distributed file systems. Hence it solves the analytical functionality it currently lacks. In Apache foundation having more focus to satisfy their customers. The searching of data in Hadoop Distributed File System (HDFS) is not yet integrated with Hadoop framework, hence in this paper, proposed a new Hadoop Search (Hasear) tool with the existing Hadoop framework. This searching algorithm can give good performance with the traditional search algorithms, it can also search with various file formats and even current mailing services data. The proposed Hasear tool is integrated with Hadoop 2.7.2 version, to search the data in the distributed file system in various virtual or real machines.

REFERENCES

1. Pal, A., K. Jain, P. Agrawal and S. Agrawal, 2014. A Performance Analysis of MapReduce Task with Large Number of Files Dataset in Big Data Using Hadoop,in Communication Systems and Network Technologies (CSNT), 2014 Fourth International Conference on, pp: 587-591.
2. Singh, K. and R. Kaur, 2014. Hadoop: Addressing challenges of Big Data, in Advance Computing Conference (IACC), 2014 IEEE International, pp: 686-689.
3. Manikandan, S.G. and S. Ravi, 2014. Big Data Analysis Using Apache Hadoop, in IT Convergence and Security (ICITCS), 2014 International Conference on, pp: 1-4.
4. Patel, A.B., M. Birla and U. Nair, 2012. Addressing big data problem using Hadoop and Map Reduce,in Engineering (NUiCONE), 2012 Nirma University International Conference on, pp: 1-5.
5. Kotiyal, B., A. Kumar, B. Pant and R.H. Goudar, 2013. Big data: Mining of log file through hadoop, in Human Computer Interactions (ICHCI), 2013 International Conference on, pp: 1-7.
6. Rama Satish, K.V. and N.P. Kavya, 2014. Big data processing with harnessing hadoop - MapReduce for optimizing analytical workloads, in Contemporary Computing and Informatics (IC3I), 2014 International Conference on, pp: 49-54.
7. Saldhi, A., A. Goel, D. Yadav, A. Saldhi, D. Saksena and S. Indu, 2014. Big data analysis using Hadoop cluster, in Computational Intelligence and Computing Research (ICCIC), 2014 IEEE International Conference on, pp: 1-6.
8. Mazumdar, S. and S. Dhar, 2015. Hadoop as Big Data Operating System -- The Emerging Approach for Managing Challenges of Enterprise Big Data Platform, in Big Data Computing Service and Applications (BigDataService), 2015 IEEE First International Conference on, pp: 499-505.