MapReduce Programs Simplification using a Query Criteria API

Boulchahoub Hassan, Khalil Namir, Amina Rachiq, Labriji Elhoussin, Benabbou Fouzia
Department of Mathematics and Computer Science
Faculty of Sciences Ben M'SIK
Casablanca, Morocco

***Abstract*—A Hadoop HDFS is an organized and distributed collection of files. It is created to store a huge part of data and then retrieve it and analyze it efficiently in a less amount of time. To retrieve and analyze data from the Hadoop HDFS, MapReduce Jobs must be created directly using some programming languages like Java or indirectly using some high level languages like HiveQL and PigLatin. Everyone knows that creating MapReduce programs using programming languages is a difficult task that requires a remarkable effort for their creation and also for their maintenance. Writing MapReduce code by hand needs a lot of time, introduce bugs, harm readability, and impede optimizations. Profiles working in the field of big data always try to avoid hard and long programs in their work. They are always looking for much simpler alternatives like graphical interfaces or reduced scripts like PIG Latin or even SQL queries. This article proposes to use a MapReduce Query API inspired from Hibernate Criteria to simplify the code of MapReduce programs. This API proposes a set of predefined methods for making restrictions, projections, logical conditions and so on. An implementation of the Word Count example using the Query Criteria API is illustrated in this paper.**

***Keywords*—Hadoop; HDFS; MapReduce**

I. Introduction

Big data analysis has become a priority for all companies and organizations that want to maintain a high level of competition. To accomplish this task, companies use several frameworks like Hadoop ecosystem which ensures both storage and processing despite the huge volume of data. Hadoop [1], [2] contains mainly a distributed File System HDFS [3] and a distributed computation framework MapReduce [4].

To analyse data stored in HDFS and according to the user's profile and competence, several programming languages are used such as java to create MapReduce programs directly [5]. Some high level languages are also used like PIG Latin scripts or HiveQL queries [6]. In this domain, several research projects have attempted to simplify the code of MapReduce programs to make them readable and easily maintainable.

This article suggests using an API called MapReduce Criteria inspired from the Hibernate Criteria API to hide the code of the restrictions and projections made on the data stored in the HDFS. Thus, the number of lines in MapReduce programs will be reduced to facilitate readability and maintenance.

This paper is organized as follows. Section 2 describes Hadoop ecosystem. Section 3 presents the motivation of using MapReduce Criteria. As for Section 4, it talks about Pig, Hive and Sqoop as related work. Section 5 briefly describes the Query Criteria API. The last section contains final conclusions and points to further work.

II. Hadoop Ecosystem

Hadoop ecosystem is a set of popular frameworks [7] that provides distributed processing over a huge amount of data. Hadoop is designed to solve data storage problems caused by the large amount of data generated each second. The data managed by this framework is processed in a parallel way by exploiting thousands of machines. Fig. 1 shows a simple architecture of Hadoop ecosystem.

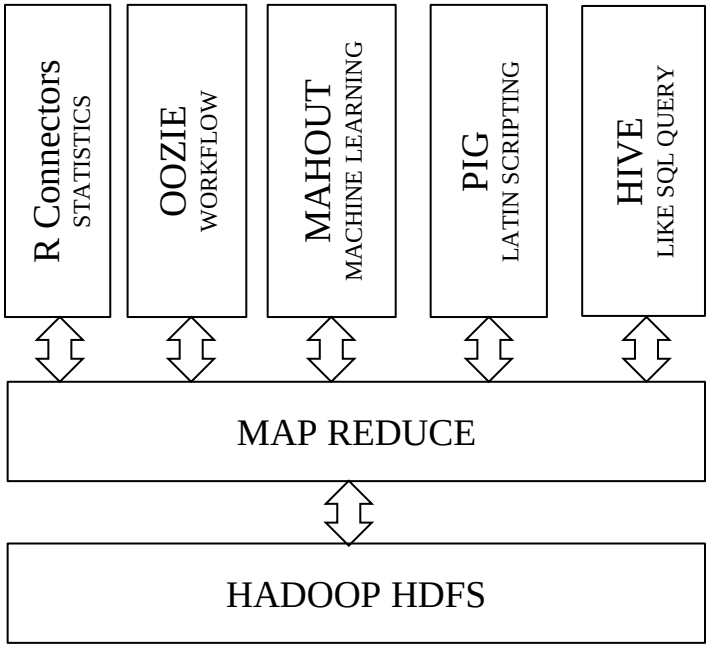

Fig. 1. Simple architecture of Hadoop ecosystem.

Along with the market trends and the diversity of profiles that intervene on the data, several additional technical components have been emerged; components for people who

are used to work in declarative SQL language and other components that are based on procedural languages.

With the multitude of solutions that currently exist in the market, each profile must carefully choose the Big Data solution that aligns with its skills. The analysts will likely find that they can ramp up on Hadoop faster by using Hadoop data warehouses such as Hive [8], Impala [9] and HAWQ now frequently deployed at customer sites. Developers who want better control of the data flow process and those who come from a procedural language context will choose to work in PIG Latin. Despite the diversity of existing solutions, they all use the same HDFS for cluster storage and the same MapReduce model for distributed processing (Fig. 2).

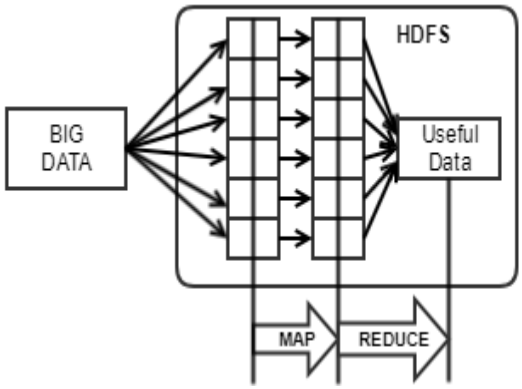

Fig. 2. Conceptual overview of HDFS and MapReduce.

A. Hadoop Distributed File System (HDFS)

To meet the ever-changing volumes of data processed every day, The Hadoop Distributed File System (HDFS) is designed to be highly fault-tolerant and to be deployed on low-cost hardware. HDFS is based on a master / slave architecture. It offers a master server (NameNode) and slaves (DataNodes) per node of the cluster [3]. The NameNode manages the namespace of the file system and also orchestrates access to the files by the clients. The DataNodes manages the storage associated with the nodes on which they run. A simple HDFS architecture is given at Fig. 3.

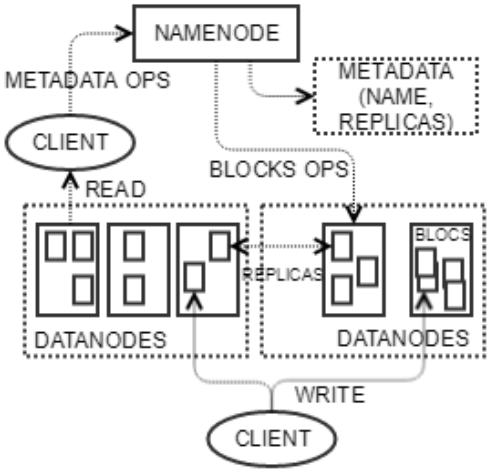

Fig. 3. Architecture of the HDFS.

B. MapReduce

The basis of the MapReduce framework was defined by Dean and Ghemawat at their paper in 2004 [5]. MapReduce orchestrates the processing of a large data sets using parallel computing on a cluster. It manages all issues related to partitioning the input data, scheduling the program's execution and data transfers. Several research papers are focused on the MapReduce model to apply it to some business domains [10], [11] to resolve some algorithms issues [12], [13] or to search for some optimization leads [14], [15]. The user of the MapReduce library expresses the computation as two functions: Map and Reduce.

Map function takes an input pair and produces a set of intermediating key/value pairs. It gathers together all intermediate values associated with the same intermediate key and passes them to the Reduce function. Reduce function written by the user accepts an intermediate key and a set of values for that key. It merges these values to form a possibly smaller set of values (Fig. 4).

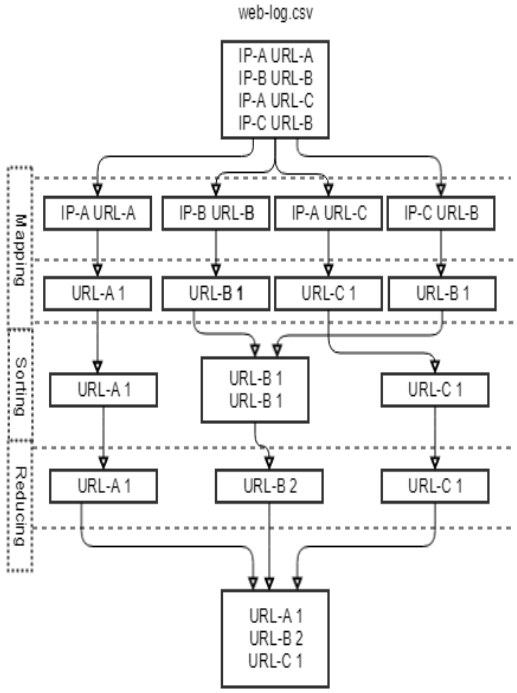

Fig. 4. Map function showing values to form a possibly smaller set of values.

III. Motivation

MapReduce programs are positioned in the core of all BigData systems. Unfortunately MapReduce programs have been criticized for several disadvantages including the large number of instructions, the lack of readability and also the difficulty of maintenance.

In order to simplify the number of instructions, the readability and the maintenance of the MapReduce programs, we propose to use the MapReduce Query API which will hide all the instructions related to:

- **Restrictions** like equal, not equal, less than, more than, etc.

- **Logical expressions like AND, OR, XOR etc.**
- **Relations between data** like "inner_join", "left_join"
- **Projections** like group, maximum, minimum, average, etc.
- **Orders** ascendant and descendant.

We propose to apply all methods defined in Hibernate Criteria [16] to MapReduce programs. Among the major concerns of big data solutions today, we can mention the optimization of execution times and also the simplification of creating MapReduce programs. Solutions mentioned at the next section try to hide the complexity of MapReduce programs by generating MapReduce plans automatically.

IV. Related Work

Different tools and sub-projects have been created to simplify the task for users who are not so good at programming languages. Many frameworks have been implemented to help users who are struggling with Hadoop, especially while performing any MapReduce tasks. Among these solutions, we find Pig, Hive and Sqoop described briefly in the following paragraphs.

A. Pig

Pig is a procedural language platform used to develop a script Pig Latin [17]: a sequence of steps, much like in a programming language, each of which carries out a single high-level data transformation e.g., filtering, grouping, or aggregation.

B. Hive

A data warehouse solution that allows users to write SQL like Query (HiveQL) and translate them into physical plans of MapReduce jobs using the Thrift Server. Hive proposes many external interfaces (Command Line, Web UI, JDBC...) to challenge with its database [18]-[20]. The latest version of Hive (since version 2.0) allows also procedural SQL on Hadoop [21].

C. Sqoop

This solution is also adopted by the Apache Foundation in order to achieve bulk data transfers between Hadoop and structured databases such as relational databases. Sqoop hides and simplifies the complexity of MapReduce programs to users [22], [23].

V. Proposed Work

The MapReduce Query API inspired from Hibernate Criteria API will represent a query against a particular file stored at the HDFS. The interface will provide the same powerful mechanism of hibernate criteria API and will allow a programmatic creation of queries against the HDFS (Fig. 5). It's an alternate way to manipulate objects generated from data stored at the HDFS. Specifying the structure of the data to be loaded from the HDFS is required for using MapReduce Query API. It is the equivalent of Relational Object Mapping in the Hibernate Framework. Any program based on MapReduce Query API will be automatically translated to MapReduce programs according to a previously defined plan.

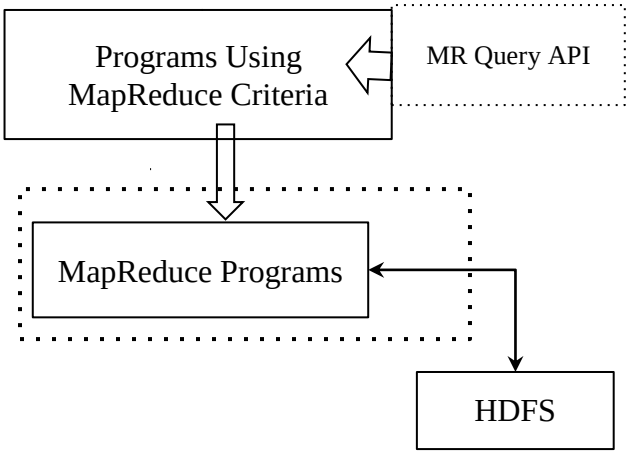

Fig. 5. MapReduce processing.

VI. Implementation

WordCount is a famous application that counts the number of occurrences for each word in a given set of files. The input for this implementation is a file of comments as detailed below:

A. WordCount Example without MapReduce Criteria

To develop a simple MapReduce example in the current model, it is necessary to create at least three classes: A class "Mapper" as shown in Table I, a "Reduce" class as shown in Table II and a "Main" class as shown in Table III.

TABLE I. WordCount Mapper Class

```
package com.hadoop.mapreduce.example;
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reporter;
public class WordCountMapper extends MapReduceBase
implements Mapper<LongWritable, Text, Text,
IntWritable> {
        private final static IntWritable one = new
IntWritable(1);
private Text word = new Text();
public void map(LongWritable key, Text value,
OutputCollector<Text, IntWritable> collector, Reporter
reporter) throws IOException {
String line = value.toString();
StringTokenizer st = new StringTokenizer(line, " ");
while (st.hasMoreTokens()) {
```

```
            word.set(st.nextToken().trim());
if (!"Apache".equals(word))
{
  collector.collect(word, one);
                        }
            }
}
}
```

TABLE II. WORDCOUNT REDUCER CLASS

```
package com.hadoop.mapreduce.example;
import java.io.IOException;
import java.util.Iterator;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;
public class WordCountReducer extends MapReduceBase
implements Reducer<Text, IntWritable, Text, IntWritable>
{
public void reduce(Text key, Iterator<IntWritable> values,
OutputCollector<Text, IntWritable> outputCollector,
Reporter reporter) throws IOException {
        int sum = 0;
         while (values.hasNext()) {
            sum = sum + values.next().get();
          }
outputCollector.collect(key, new IntWritable(sum));
}
}
```

TABLE III. WORDCOUNT MAIN CLASS

```
package com.hadoop.mapreduce.example;

import java.net.URI;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.RunningJob;
import org.apache.hadoop.mapred.TextOutputFormat;
public class WordCount {
public static void main(String[] args) throws Exception {
Configuration conf = new Configuration();
Path inputPath = new Path(
          "hdfs://127.0.0.1:9000/input/comments.txt");
Path outputPath = new
Path("hdfs://127.0.0.1:9000/output/");
JobConf job = new JobConf(conf, WordCount.class);
job.setJarByClass(WordCount.class);
job.setJobName("WordCounterJob");
FileInputFormat.setInputPaths(job, inputPath);
FileOutputFormat.setOutputPath(job, outputPath);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);
job.setOutputFormat(TextOutputFormat.class);
job.setMapperClass(WordCountMapper.class);
job.setReducerClass(WordCountReducer.class);
FileSystem hdfs =
FileSystem.get(URI.create("hdfs://127.0.0.1:9000"),
                                        conf);
if (hdfs.exists(outputPath))
hdfs.delete(outputPath, true);
RunningJob runningJob = JobClient.runJob(job);
System.out.println("job.isSuccessfull: " +
runningJob.isComplete());
}
}
```

B. WordCount Example using MapReduce Criteria

The objective of MapReduce Criteria is to reduce the number of rows and classes for developers using MapReduce; it will facilitate the creation and also the maintenance of their programs. Each program based on MapReduce Criteria will be automatically translated into Mappers and Reducers classes. The example "WordCount" in MapReduce Criteria is given in Table IV.

TABLE IV. WORDCOUNT EXAMPLE USING MAPREDUCE CRITERIA

```
package com.hadoop.mapreduce.example;
public class WordCountMrCriteria extends MrQueryScript
{
public static void main(String[] args) {
String hdfs = "hdfs://127.0.0.1:9000";
DataList dlist1 = load(hdfs + "/input/comments.txt",
new String[] {"line:String"});
DataList dlist2 = null;
for (DataObject dobj : dlist1.getDataObjectList()) {
dlist2 = tokenize(dobj,"line", " ", new String[]
{"word:String"});
}
MapReduceCriteria c1 = dlist2.getMrCriteria()
.add(Restrictions.ne("word", "Apache"))
.add(Projections.groupBy("word"))
.add(Projections.rowCount());

DataList dlist3 = c1.dataList();
dlist3.store(hdfs + "/output/countword.txt");
}
}
```

VII. Conclusion

MapReduce is a programming model created to perform distributed processing of a large datasets stored in a distributed file system HDFS. It is well-known that MapReduce programs are difficult to create, to read and to maintain. Therefore, it is necessary to simplify them using some frameworks or APIs.

This paper has suggested using an API called MapReduce Criteria in order to reduce the number of MapReduce instructions and also to hide Mappers and Reducers classes for developers. In our future work we will compare MapReduce programs that use the Query Criteria API with existing languages that also simplify the use of MapReduce as Pig Latin and Hive.

Acknowledgment

We would like to thank Mr Labriji and Mr Rachiq for their useful comments and their dedicated work.

References

[1] D. Keulen, Hadoop: The Definitive Guide. CreateSpace, 2014.

[2] "Welcome to ApacheTM Hadoop®!" [Online]. Available: http://hadoop.apache.org/.

[3] "HDFS Architecture Guide." [Online]. Available: https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html.

[4] "Apache Hadoop 2.9.0 – MapReduce Tutorial." [Online]. Available: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html

[5] J. Dean and S. Ghemawat, "MapReduce," Commun. ACM, vol. 51, no. 1, 2008, p. 107.

[6] S. Stewart, The Hive. University of Georgia Press, 2008.

[7] "Hadoop Ecosystem: An Introduction," International Journal of Science and Research (IJSR), vol. 5, no. 6, 2016, pp. 557–562.

[8] E. Capriolo, D. Wampler, and J. Rutherglen, Programming Hive. "O'Reilly Media, Inc.," 2012.

[9] J. Russell, Getting Started with Impala: Interactive SQL for Apache Hadoop. "O'Reilly Media, Inc.," 2014.

[10] U. D., D. Umesh, and B. Ramachandra, "Big Data Analytics to Predict Breast Cancer Recurrence on SEER Dataset using MapReduce Approach," Int. J. Comput. Appl. Technol., vol. 150, no. 7, 2016, pp. 7–11.

[11] Z. Wu, B. Mao, and J. Cao, "MRGIR: Open geographical information retrieval using MapReduce," in 2011 19th International Conference on Geoinformatics, 2011.

[12] Q. Guo, B. Palanisamy, and H. Karimi, "A Distributed Polygon Retrieval Algorithm using MapReduce," in Proceedings of the 10th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing, 2014.

[13] M. A. Alshammari and E.-S. M. El-Alfy, "MapReduce implementation for minimum reduct using parallel genetic algorithm," in 2015 6th International Conference on Information and Communication Systems (ICICS), 2015.

[14] Q. Liu, W. Cai, B. Wang, Z. Fu, and N. Linge, "An Optimization Scheme in MapReduce for Reduce Stage," International Journal of Grid and Distributed Computing, vol. 9, no. 8, 2016, pp. 197–208.

[15] Y. Tao, Q. Zhang, L. Shi, and P. Chen, "Job Scheduling Optimization for Multi-user MapReduce Clusters," in 2011 Fourth International Symposium on Parallel Architectures, Algorithms and Programming, 2011.

[16] "Chapter 15. Criteria Queries." [Online]. Available: https://docs.jboss.org/hibernate/orm/3.3/reference/en/html/querycriteria.html.

[17] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins, "Pig latin," in Proceedings of the 2008 ACM SIGMOD international conference on Management of data - SIGMOD '08, 2008.

[18] A. Thusoo et al., "Hive," Proceedings VLDB Endowment, vol. 2, no. 2, 2009, pp. 1626–1629.

[19] E. Capriolo, D. Wampler, and J. Rutherglen, Programming Hive. "O'Reilly Media, Inc.," 2012.

[20] J. Venner, Pro Hadoop. 2009.

[21] "HPL/SQL Reference - HPL/SQL - Procedural SQL on Hadoop, NoSQL and RDBMS," 04-Aug-2017. [Online]. Available: http://www.hplsql.org/doc.

[22] "Sqoop User Guide (v1.4.0-incubating)." [Online]. Available: https://sqoop.apache.org/docs/1.4.0-incubating/SqoopUserGuide.html.

[23] K. Ting and J. J. Cecho, *Apache Sqoop Cookbook: Unlocking Hadoop for Your Relational Database*. "O'Reilly Media, Inc.," 2013.