International Journal of Computing and Digital Systems
ISSN (2210-142X)
Int. J. Com. Dig. Sys. 10, No.1 (May-2021)

http://dx.doi.org/10.12785/ijcds/100160

Using Model Driven Engineering to Transform Big Data Query Languages to MapReduce Jobs

Allae Erraissi

Laboratory of Information Technology and Modeling, Hassan II University, faculty of sciences Ben M'Sik, Casablanca, Morocco

Received 30 May. 2020, Revised 09 Apr. 2021, Accepted 15 Apr. 2021, Published 02 May. 2021

Abstract: Big Data processing is done by using MapReduce which is a clustered data processing framework. As it is Composed of Map and Reduce functions, it distributes data processing tasks between different computers, and hence reduces the results in a single summary. Most data analysts prefer to use query languages like Pig and Hive to process Big Data, given the complexity of the MapReduce paradigm. In this paper, we shall propose an approach based on Model Engineering to transform requests written by Pig or Hive to MapReduce jobs thanks to the use of the ATL transformation language. Our proposal will allow us to easily obtain MapReduce programs from requests written in Pig or Hive.

Keywords: MapReduce, Model Driven Engineering, Hive, Pig.

1. INTRODUCTION

Big Data is a generic term used to describe the strategies and technologies used to collect, organize, process, and analyze large data sets. It is the art of managing and exploiting large volumes of data [1]. To process this large amount of data, we find the Hadoop ecosystem [2] mostly as Hadoop remains the main Big Data platform used today. Seeing that it is based on Java, the open-source Hadoop framework is applied to store and process data in bulk. Hadoop is part of the Apache project, which is also behind the Pig, Hive, and Spark frameworks.

Hive has initially been a Facebook project that links the SQL world to Hadoop. it executes the queries written by SQL on a cluster to aggregate and analyze data. The language used for this reason is named HiveQL [3]. Only "Select" instructions are supported to manipulate data because it is a visualization language only. In some cases, developers must map between data structures and Hive. Besides that, Pig is originally a Yahoo project which allows querying Hadoop data from a scripting language [4]. Unlike Hive, Apache Pig uses Pig Latin query language that allows us to create MapReduce type programs. Pig Latin abstracts from the Java MapReduce programming language and goes to a higher level of abstraction, similar to that of SQL for RDBMS (Relational Database Management System). Unlike Hive, Pig does not have a web interface [5].

Since we know that to do Big Data processing, we will need the MapReduce paradigm [6]. So, the requests written by the HiveQL language or PigLatin transform to MapReduce jobs according to the classic Big Data architecture. During this paper, we continue the application of Model-Driven Engineering techniques to standardize concepts at the Big Data. In the previous contribution [7] we have already proposed two meta-models for Ingestion and Data sources layers. Next, we defined a meta-modeling for the other layers of Big Data which are: Storage [8,9], Visualization [10,11], and Security [12]. The work presented in this paper is an advancement report on our first proposal for a meta-modeling of the Big Data Management layer [13]. Our main goal is to provide a generic Big Data system based on Model-Driven Engineering [14] to address the problem of many Big Data solutions.

This article is detailed as follows: section 2 presents the related work on which we are concerned to carry out this study. Then, in section 3 we talk about the MapReduce paradigm used to manage Big Data. Sections 4 and 5 present Hive and Pig query languages and their modes of use. Section 6 presents our proposed approach which makes it easy to obtain MapReduce programs from queries written in Pig or Hive. Finally, section 7 presents the results and evaluation obtained after the application of our approach based on Model Driven Engineering.

E-mail: allae.erraissi-etu@etu.univh2c.ma

2. Related work

As part of our research project, we continue the application of Model-Driven Engineering techniques to standardize Big Data concepts. In the previous contribution [7] we have already proposed two meta-models for Ingestion and Data sources layers. Next, we defined a meta-modeling for the other layers of Big Data which are: Storage [8,9], Visualization [10,11], and Security [12]. The work presented in this paper is an advancement report on our first proposal for a meta-modeling of the Big Data Management layer [13].

The main paradigm used to process Big data is MapReduce [6], plus other complementary tools like Pig [4], Hive [3], Sqoop [15], etc. In this paper, we shall treat the two query languages, Pig and Hive, which aim to process Big Data thanks to two languages dedicated to this reason which are: Pig Latin and HiveQL. Queries that are written by these two languages will be transformed to MapReduce jobs according to the classic Big Data architecture. We aim to consolidate a generic Big Data system based on Model Driven Engineering to solve the problem of many existing solutions in the market today.

Many studies have addressed the problem of the cost of data transfers within MapReduce applications. Most of them deal with the locality of the data during the map phase. One of the algorithms proposed [16] improves this locality by introducing a delay before migrating a task to another node, if the preferred node is not available. The BAR [17] algorithm aims to approach the optimal data distribution taking into account an initial configuration that will be dynamically adapted.

LEEN [18] is an algorithm for partitioning intermediate keys that aim to balance the duration of reduction while trying to reduce bandwidth consumption in the shuffle. LEEN is based on the frequency of the intermediate appearance of keys in an attempt to create partitions and optimize data transfers. The HMPR algorithm [19] defines a pre-shuffling mode which tends to reduce the data quantity to be transferred. For this reason, it predicts the partition in which the data will be generated at the output of the map and has the piece of data processed by the node which will execute the Reduce of this partition if possible.

The Ussop environment [20] permits to target of the grids and adapt the quantity of data to be treated by each map according to the computing power of the machine running it. Also, this tool tends to reduce intermediate data transfers by locally performing the Reduce on the machine that generated the most intermediate keys.

A MapReduce program splits the work we want to do, into a set of tasks that will be delivered to map functions. It is therefore possible to use results from the theory of divisible tasks [21] for this kind of application. This approach was introduced by Drozdowski and Berlinska [22]. In this article, the authors consider an execution environment in which number of compute nodes is bigger than number of communications that can take place simultaneously without causing contention. To avoid the appearance of this phenomenon, the authors propose a new model that permits the run of a MapReduce job by a linear program that generates a distribution of the data and static scheduling by phases of the communications. If this approach turns out to be interesting, the use of a linear program makes it inapplicable for instances involving more than a few hundred maps because the resolution time can sometimes exceed several minutes. Also, it happens that the linear program solver fails for such instances. The sequencing by phases induces, in addition, a large number of idle times on machines and the network during the shuffle.

3. MapReduce

The MapReduce paradigm [6] was presented in 2008 by Dean et al [23]. In a MapReduce program, two types of operations are chained to perform a calculation: The Map operation and the Reduce operation. All these operations form a MapReduce job.

In the MapReduce paradigm, data is represented by key-value pairs. The Map and Reduce operations take as input a set of key-value pairs and return a set of key-value pairs. We use the operator to describe the power set (i.e., the set of parts) of a set. For a job, the Map and Reduce functions are written:

$$
\begin{gathered}
map : K \times V \rightarrow \beta(K' \times V') \\
(k, v) \rightarrow \{(k', v') | (k', v') \in K' \times V'\} \\
reduce : K' \times \beta(V') \rightarrow \beta(K'' \times V'') \\
(k', \{v' | v' \in V'\} \rightarrow \{(k'', v'') | (k'', v'') \in K'' \times V''\}
\end{gathered}
$$

These operations are designed to allow easy distribution of the calculations. On a computing cluster, each machine processes only part of the data, to take advantage of distributed file storage.

A. Map operation

The Map operation transforms the input data to an intermediate state usable by the Reduce operation [24]. Depending on the application, it can be used to filter data, duplicate data, etc. In the distributed implementation of the paradigm, the machines performing the Map operation are called mappers. Each mapper then only works on part of the data. The key to the element's output from the Mapper function is to determine the reduced values together in the Reduce operation.

B. Reduce operation

The Reduce operation transforms values with the same key into a key-value pair. Depending on the application, the Reduce operation transforms the output of the Map into a statistical indicator, into a sorted data set, etc. In the distributed implementation of the paradigm, machines performing the Reduce operation are called a reducer. Each reducer works on a part of the data [25].

C. *Shuffle of data*

Between the Map operation and the Reduce operation, the dataset undergoes a modification. The Map operation returns a set of key-value pairs and the Reduce operation takes a key-set of values as input. The Shuffle step allows for this transition [26]. During the data shuffle, the data output from the Map operation is transmitted over the network towards the reducers. Pairs with the same keys are transmitted to the same reducer. Key-set pairs of values are formed this way. This transfer can represent an important blocking point for the distributed implementation. When handling large datasets, the amount of information exchanged over the network can be significant (up to several times the size of the datasets, depending on the application). It is therefore necessary to pay attention to the volume of data transmitted during this step to improve the performance of the algorithm.

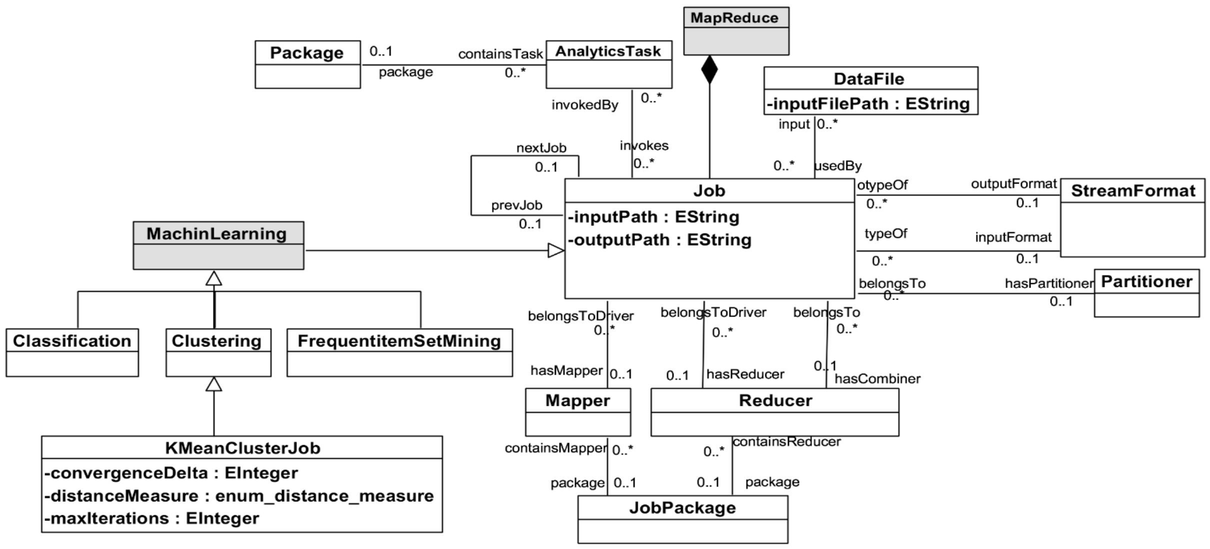

Figure 1. Proposed MapReduce meta-model.

4. PIG

Pig is a big data processing tool [4]. This software is created by Yahoo. Pig offers to perform processing on distributed data sets using a natural programming language, much like SQL. The query language used to manipulate big data with the Apache Pig tool is Pig Latin [28]. The goal of using Pig Latin to define simple scripts, such as SQL, is to make it easier to write large data processing programs without having to write Java jobs through the MapReduce framework. The Pig solution was adopted by the Apache organization and is now an essential component of the Hadoop suite [27].

A. *Example of a Pig program*

This program displays the 11 youngest adults extracted from a csv file containing 3 columns: identifier, name, and age.

```
persons = LOAD ‘persons.csv’ USING PigStorage(‘;’)
AS (userid:int, name:chararray, age:int);
youngadults= FILTER persons BY age>= 18 AND age <25;
ranking = ORDER youngadults BY age;
result = LIMIT ranking 11;
DUMP result;
```

To run it: freelance program.pig. It's launching a MapReduce job in Hadoop. You can also type the instructions one by one into Pig's shell.

B. *Comparison between SQL and Pig Latin*

There are some apparent similarities between SQL [29] and Pig Latin. There are several keywords in common (JOIN, ORDER, LIMIT, etc.) but their principle is different:

- In SQL, queries are built that describe the data to be obtained. It is not known how the SQL engine will calculate the result. We only know that internally, the query will be broken down into loops and in comparison, on the data and making the best use of the indexes.
- In Pig Latin, programs are built that contain instructions. It describes exactly how the result should be obtained, what calculations should be made, and in what order.

Also, Pig was designed for uncertain Hadoop data, while SQL runs on perfectly healthy RDBMS.

C. *Pig Latin Language*

1) *Structure of a program*

Comments are placed between /*...*/ or from - and the end of the line. A Pig Latin program is a series of instructions. All must be terminated with a; As in SQL, there is no notion of variables, nor functions/procedures. The result of each Pig statement is a collection of tuples.

We call it a relationship. We can see it as a database table. Each Pig instruction takes an input relation and produces a new output relation.

output - INSTRUCTION input PARAMETRES...;

2) Running a program

When you run a program, Pig first analyzes it. Each instruction, if it is syntactically correct, is added to a kind of action plan, a succession of MapReduce, and it is only at the end of the program that this action plan is executed according to what you ask at the end.

The EXPLAIN relation instruction displays the action plan planned to calculate the relation. It's pretty indigestible when you're not a specialist.

3) Relationships and aliases

The syntax name = INSTRUCTION...; defines an alias, i.e., a name for the relation created by the instruction. This name is generally used in the following instructions, this is what builds a processing flow.

```
name1 = LOAD ... ;
name2 = FILTER name1... ;
name3 = ORDER name2... ;
name4 = LIMIT name3... ;
```

The same alias can be reused in different instructions, which creates bifurcations in the processing flow: separations or groupings. It is not recommended to reassign the same alias.

4) Chaining of instructions

Pig allows you to either chain instructions through the alias mechanism, or through a nested call.

```
name4 = LIMIT (ORDER (FILTER (LOAD ...) ...) ...) ... ;
```

You will choose the one you find most readable. However, nested calls do not allow easy separation of processing, unlike aliases:

```
name1 = LOAD ... ;
name2 = FILTER name1 ... ;
name3 = FILTER name1 ... ;
name4 = JOIN name2 ..., name3 ;
```

5) Relationships and types

A relation is an ordered collection of tuples that all have the same fields. Here are the possible types. The scalar types are:

- int and long for integers, float and double for reals
- chararray for any chains.
- bytearray for any binary objects
- There are also three complex types:
- dictionaries (maps): [name # mickey, age # 87]
- tuples of fixed size: (mickey, 87, hergé)
- sacs (bags) = sets without tuples order: {(mickey, 87), (asterix, 56), (tintin, 86)}

6) Schema of a relationship

The list of fields in a relationship is called a schema. It is a tuple. We write it (name1: type1, name2: type2, ...).

For example, a relationship containing employees will have the following schema:

```
(id:long, lastname:chararray, firstname:chararray,
picture:bytearray, seniority:int, salary:float)
```

The LOAD instruction 'file.csv' AS diagram; allows to read a CSV file and to make a relation according to the indicated diagram.

7) Complex schema (tuples)

Pig allows the creation of a relationship based on a diagram including complex data. Either a file containing 3D segments:

```
S1 ➪ (3,8,9) ➪ (4,5,6)
S2 ➪ (1,4,7) ➪ (3,7,5)
S3 ➪ (2,5,8) ➪ (9,5,8)
```

We use the character to represent a tabulation. Here is how to read this file:

```
segments = LOAD 'segments.csv' AS (
            name:chararray,
            T1:tuple(a1:int, b1:int, c1:int),
            T2:tuple(a2:int, b2:int, c2:int));
DUMP segments;
```

8) Complex schema (bags)

We can also read bags, i.e., data sets of the same types but in any number:

```
L1 ➪ {(3,8,9),(4,5,6)}
L2 ➪ {(4,8,1),(6,3,7),(7,4,5),(5,2,9),(2,7,1)}
L3 ➪ {(4,3,5),(6,7,1),(3,1,7)}
```

The diagram of this file is:

```
(name:chararray, Points:{tuple(a:int, b:int, c:int)})
```

Explanations:

- The second field of the diagram is specified as "field name": "type of bag content".
- Data in this field should be in the "list of values for the type" format.

9) Complex schema (maps)

Finally, with dictionaries, here are the contents of the heros.csv file:

```
1 ➪ [name#asterix, job#warrior]
2 ➪ [name#tintin, job#journalist]
3 ➪ [name#spirou, job#groom]
```

It is made a relationship by:

```
heros = LOAD 'heros.csv' AS (id:int, info:map[chararray])
```

Note: All these constructions, map, bags, and tuple, can be nested, but some combinations are difficult to specify.

10) Field names

There are two syntaxes for naming the fields of a relationship. Either use their plain name or designate them by their position $0 designating the first field, $1 the second, and so on.

The second syntax is used when the names of the fields are not known or when they have been generated dynamically.

When there is ambiguity on the relation concerned, we prefix the name of the field with the name of the relation: relation.champ.

- When a field is a tuple, its elements are named relation.champ.element. For example segments.P1.z1
- For a map type field, its elements are named relation.champ # element. For example heros.infos # metier
- There is no syntax for accessing bag fields.

The following figure shows our proposition for Pig, it defines our meta-model:

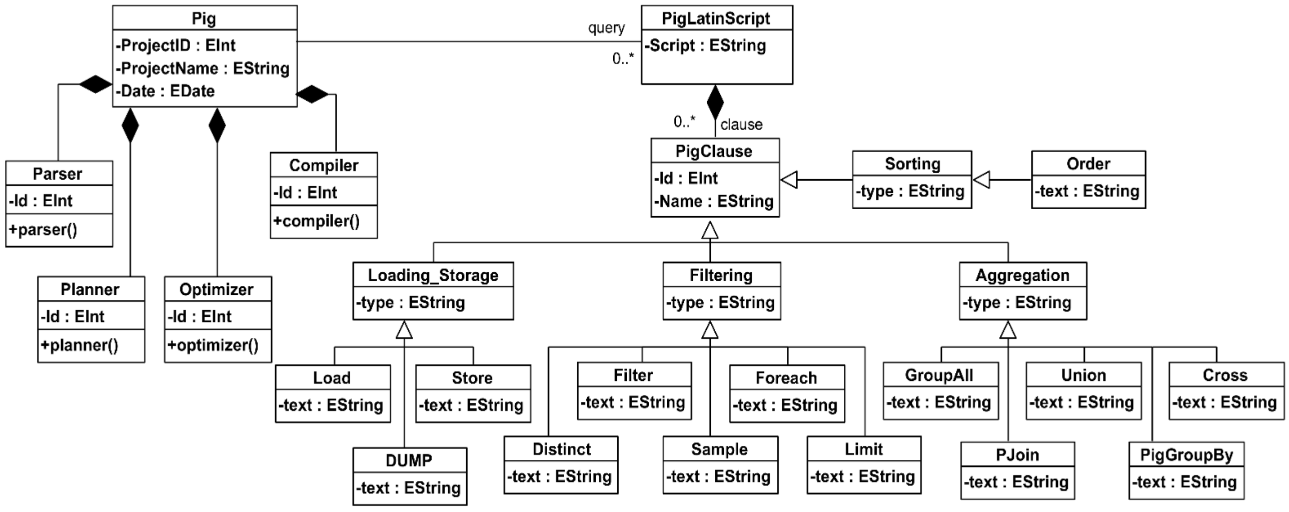

Figure 2. Meta-model of PIG.

5. HIVE

Apache Hive is a data warehouse for Hadoop like HBase [30]. It was created by Facebook to later become an open-source Apache project. This is a system that maintains metadata describing data stored in HDFS. It uses a relational database called metastore (Derby by default) to ensure metadata persistence [31]. Thus, a table in Hive is essentially composed of a schema stored in the metastore, and data stored in HDFS. With the metastore data, Hive allows manipulating the data as if they were persisted in tables (in the sense of a classic database management system) and to query them with its HiveQL language [32]. Hive converts HiveQL queries into MapReduce or Tez jobs (from version 0.13 of Hive, a HiveQL query can be translated into an executable job on Apache Tez, which is an execution framework on Hadoop that can replace MapReduce).

A. Defining a diagram

The diagram of a table is also called metadata (i.e., data information). Metadata is stored in a MySQL database, called metastore. Here is the definition of a table with its diagram:

```
CREATE TABLE statement (
            idstatement STRING,
            year INT,
            temp FLOAT, quality BYTE, ...)
ROW FORMAT DELIMITED FIELDS TERMINATED BY
'\t';
```

The beginning is classic, except for the constraints of integrity: there are none. The end of the query indicates that the data is in a CSV file. Let us first look at the types of columns.

B. HiveQL Types

Hive defines the following types [33]:

- BIGINT (8 bytes), INT (4), SMALLINT (2), BYTE (1 byte).
- FLOAT and DOUBLE.
- BOOLEAN worth TRUE or FALSE.
- STRING, we can specify coding (UTF8 or other).
- TIMESTAMP is expressed in a number of seconds. Nanoseconds since 01/01/1970 UTC.

- structured data as with Pig:
  - ARRAY indicates that there is a list of types.
  - STRUCT for a multi-value structure.
  - MAP for a suite of. key (pairs, value).

C. *Field Separations for Reading*

The creation of a table is done as follows:

```
CREATE TABLE name (schema) ROW FORMAT
DELIMITED descr format
```

The guidelines after the diagram indicate how the data is stored in the CSV file. These are:

- FIELDS TERMINATED BY ';': there is one; to separate the fields.
- COLLECTION ITEMS TERMINATED BY ',': there is a, between the elements of an ARRAY.
- MAP KEYS TERMINATED BY ':': there is one: between the keys and values of a MAP.
- LINES TERMINATED BY 'n': there is a 'n' at the end of the line.
- STORED AS TEXTFILE: It is a CSV.

D. *Loading data*

Here is how to load a CSV file that is on HDFS [34] in the table:

```
LOAD DATA INPATH '/share/allae/data/183'
OVERWRITE INTO TABLE statement;
```

You can also load a local file (not HDFS):

```
LOAD DATA LOCAL INPATH 'stations.csv'
OVERWRITE INTO TABLE stations;
```

The file is then copied to HDFS in Hive's files.

E. *HiveQL requests*

As with conventional RDBMS, there is a shell launched by the hive command. This is where SQL queries are typed. They are mainly SELECT. All the clauses you know are available: FROM, WHERE, JOIN, HAVING, GROUP BY, LIMIT, ORDER BY.

There are others to optimize the underlying MapReduce work, for example when you want to rank on a column, you have to write:

```
SELECT... DISTRIBUTE BY colonne SORT BY colonne;
```

The directive sends the affected n-uplets on a single machine to compare them more quickly to establish the ranking.

F. *Other guidelines*

It is also possible to export results in a file:

```
INSERT OVERWRITE LOCAL DIRECTORY
'/tmp/weather/hot'
SELECT year,month,day,temperature
FROM statement
WHERE temperature > 40.0;
```

Other orders include:

- SHOW TABLES; to view the list of tables (they are in the metastore).
- DESCRIBE EXTENDED table; shows the table schematic.

G. *Hive meta-model*

The following figure shows our proposition for Hive, it defines our meta-model for Hive and its HiveQL query language:

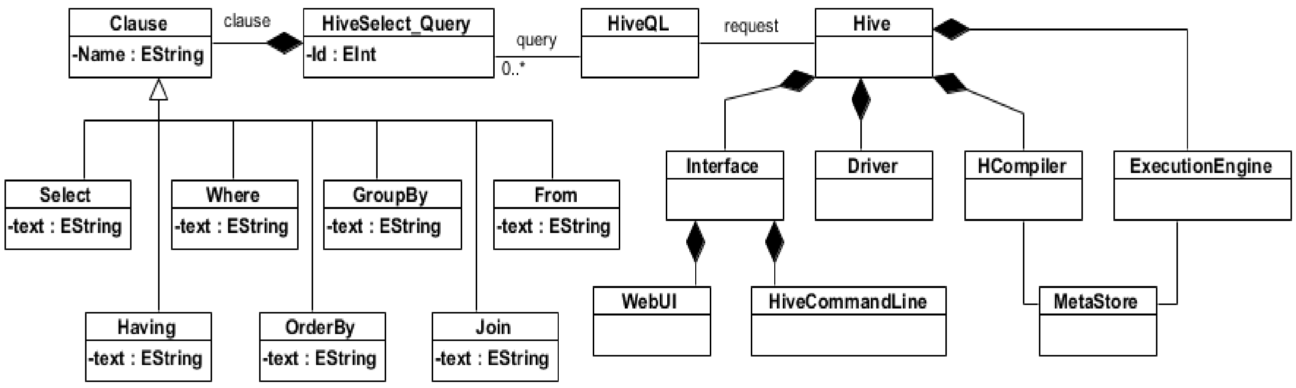

Figure 3. Hive meta-model.

6. Transformation

After defining meta-models for MapReduce, Pig, and Hive. In this section, we present the ATL transformation rules used to pass from our generic meta-models of PIG and Hive query languages to MapReduce meta-model which represents the body of processing within Big Data. The following figure shows the architecture of our proposal.

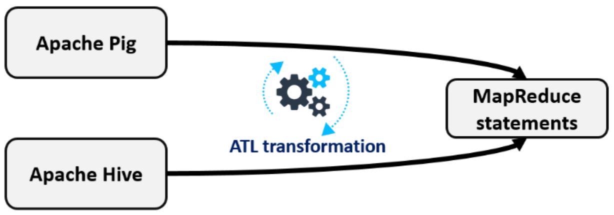

Figure 4. Architecture of PigHive2MapReduce.

To apply the transformations defining in figure 4, we have chosen the ATL transformation language [35,36]. We now present the ATL transformation rules used to pass from our generic meta-models of PIG and Hive query languages to MapReduce meta-model which represents the body of processing within Big Data. These defined meta-models present the PIM level according to the Model Driven Architecture "MDA" [41,37].

```
rule Pig2MapReduce{
      from
    s: MegaModelHadoopManagement!Pig
      to
    t: MegaModelHadoopManagement!MapReduce(
          inputPath<- s.Pig.inputPath,
                        outputPath<-          new
                        outputPath(C:\Users\AllaeE
                        rraissi\Desktop\
                        PigHive2MapReduceFolder\
                        PigOutput)
          IdMap<- s.Pig.ProjectID,
          Mapper<-s. PigLatinScript.PigClause,
          Reducer<- s.Pig.Complier,
               InputFilePath<- s.InputFilePath
               JobPackage<-s.Pig.Project,
               MapReuce.Date<-s.Pig.Date,
               query<- s.Pig.PigLatinScript
        )
}
rule Hive2MapReduce{
      from
        s: MegaModelHadoopManagement!Hive
      to
        t: MMClouderaManagement!Cldr_Hive(
                                  inputPath<-
          s.Hive.inputPath,
                        outputPath<-          new
                        outputPath(C:\Users\AllaeE
                        rraissi\Desktop\
                        PigHive2MapReduceFolder\
                        HiveOutput)
          IdMap<- s.Hive.ProjectID,
          Mapper<-
          s.Hive.HiveQL.HiveSelect_Query.Clause,
          Reducer<- s.Hive.HComplier,
               InputFilePath<- s.InputFilePath
               JobPackage<-s.Hive.Project,
               MapReuce.Date<-s.Hive.Date,
               query<-
               s.Hive.HiveQL.HiveSelect_Query
        )
}
```

7. EVALUATION AND DISCUSSION

To evaluate our approach, we used three datasets. On these datasets, we applied 30 queries to better measure the execution time of each query on the different datasets chosen to test our proposal.

To implement PigHive2MapReduce, we used version 3.1.1 of Hadoop, version 3.1.2 of Hive, and version 0.17.0 of Pig. We used a machine with a 2.80 GHz Intel (R) Core (TM) i7 processor. With a storage space of 2 TB and a RAM of 16 GB.

The three datasets used have the following sizes: DS1 (17GB), DS2 (15GB), and DS3 (13GB). The following table shows the loading time of the three datasets:

TABLE I. LOADING TIME DATASETS.

Dataset	*Dataset 1*	*Dataset 2*	*Dataset*
Loading time (s)	3,4	3,2	2,9

The following tables show the execution time of the 30 queries on the three datasets. Note that we tested PigHive2MapReduce with 15 queries using the PigLatin language, and 15 with the use of the HiveQL query language.

TABLE II. PIG REQUEST TRANSFORMATION TIME ON DATASET 1.

Pig query	*Q1*	*Q2*	*Q3*	*Q4*	*Q5*	*Q6*	*Q7*	*Q8*	*Q9*	*Q10*	*Q11*	*Q12*	*Q13*	*Q14*	*Q15*
PigHive2MapReduce (ms)	356	376	481	450	256	298	516	518	318	389	667	687	321	321	124

TABLE III. HIVE REQUEST TRANSFORMATION TIME ON DATASET1.

Query Hive	*Q1*	*Q2*	*Q3*	*Q4*	*Q5*	*Q6*	*Q7*	*Q8*	*Q9*	*Q10*	*Q11*	*Q12*	*Q13*	*Q14*	*Q15*
PigHive2MapReduce (ms)	456	445	234	213	765	656	231	124	764	343	545	432	535	654	344

TABLE IV. PIG REQUEST TRANSFORMATION TIME ON THE DATASET2.

Pig query	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Q13	Q14	Q15
PigHive2MapReduce (ms)	346	366	471	440	246	288	506	508	307	398	606	676	310	300	110

TABLE V. HIVE REQUEST TRANSFORMATION TIME ON THE DATASET2.

Query Hive	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Q13	Q14	Q15
PigHive2MapReduce (ms)	446	435	224	203	755	646	221	114	754	333	535	422	525	644	334

TABLE VI. PIG REQUEST TRANSFORMATION TIME ON THE DATASET3.

Pig query	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Q13	Q14	Q15
PigHive2MapReduce (ms)	340	360	468	434	243	278	500	502	304	374	649	669	309	312	114

TABLE VII. HIVE REQUEST TRANSFORMATION TIME ON THE DATASET3.

Query Hive	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Q13	Q14	Q15
PigHive2MapReduce (ms)	443	432	219	200	748	643	219	108	747	337	542	427	526	641	330

The results obtained after using our approach based on model engineering are shown in the following figures:

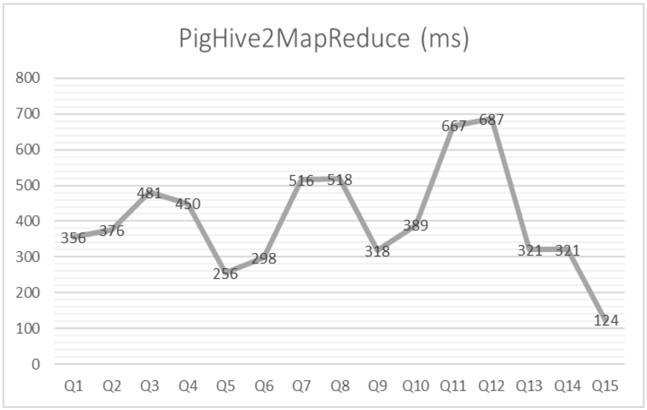

Figure 5. Transformation time of Pig requests on dataset1.

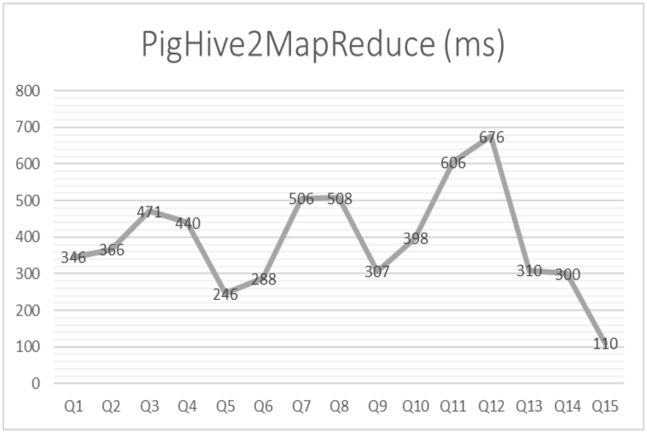

Figure 6. Transformation time of Pig requests on dataset2.

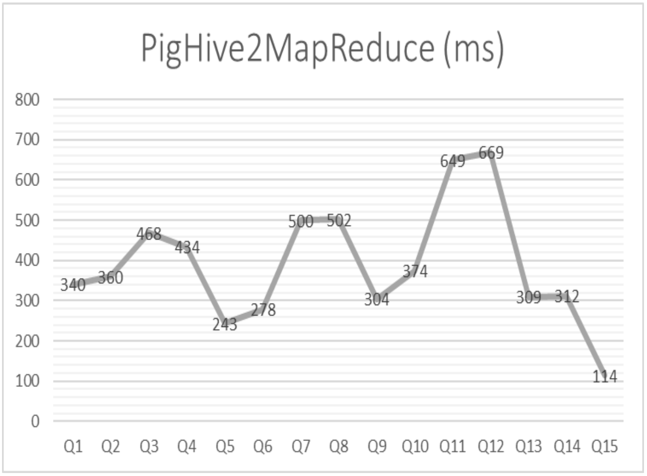

Figure 7. Transformation time of Pig requests on dataset3.

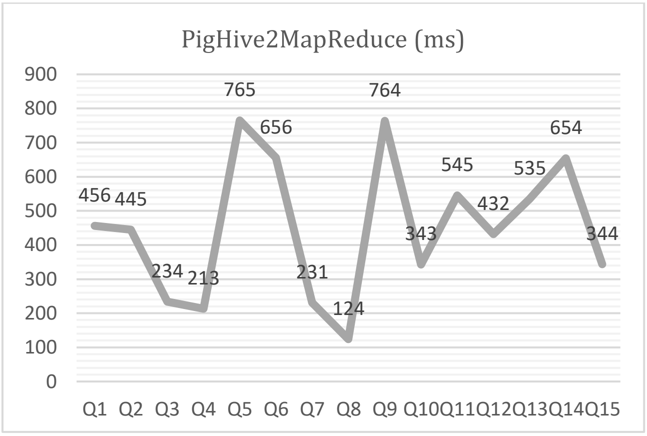

Figure 8. Transformation time of Hive requests on dataset1.

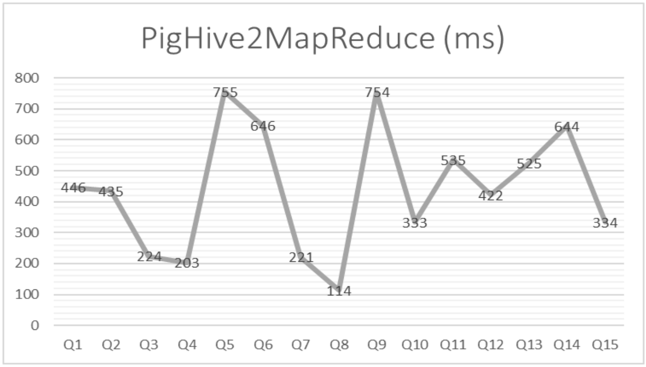

Figure 9. Transformation time of Hive requests on dataset2.

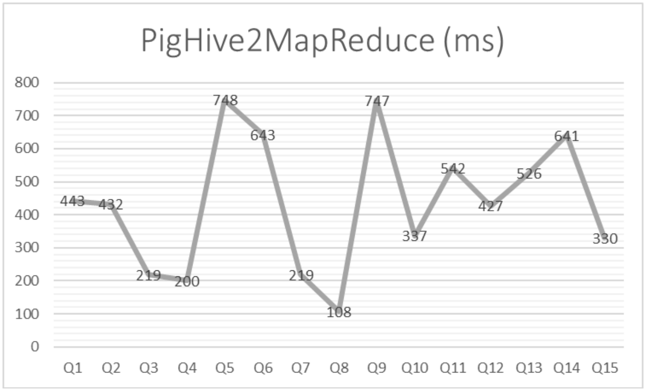

Figure 10. Transformation time of Hive requests on dataset3.

The results obtained from applying the transformations using the ATL transformation language show that the time to transform Pig Latin or HiveQL queries is speedy. The fact that will allow users of the MapReduce paradigm to obtain MapReduce programs quickly from Hive or Pig requests.

Based on the definition of meta-models for the different layers of a Big Data system in our previous work [38,40]. In continuous efforts, this work complements the meta-model already proposed for the management layer [13,39]. Precisely to define the functioning of the two Pig and Hive query languages which are based on the transformation of queries written either by the Pig Latin language or by HiveQL, into MapReduce jobs thanks to the ATL transformation language.

8. Conclusion

A large number of solutions are available in the Big Data market. Big Data solution providers have developed distributions to manage this large amount of data. However, Big Data solution providers do not have the meta-models necessary to create standard applications that can be compatible with each provider, since each provider has its policy that allows it to design its own Big Data system. So, the application of techniques related to model engineering will standardize Big Data concepts. Given the complexity of processing Big Data with the MapReduce paradigm. This paper proposes an approach based on Model Driven Engineering which makes it possible to transform the requests written by PigLatin and HiveQL to MapReduce jobs.

References

[1] Inmon, W. H., and Daniel Linstedt. "2.1 - A Brief History of Big Data." In Data Architecture: a Primer for the Data Scientist, edited by W. H. Inmon and Daniel Linstedt, 4548. Boston: Morgan Kaufmann, 2015. https://doi.org/10.1016/B978-0-12-802044-9.00008-8..

[2] Allae Erraissi, Abdessamad Belangour, and Abderrahim Tragha, "Digging into Hadoop-based Big Data Architectures," Int. J. Comput. Sci. IJCSI Issues, 14, No. 6, 52-59, Nov. 2017.

[3] Dayong Du. Apache Hive Essentials: Essential techniques to help you process, and get unique insights from, big data, 2nd Edition eBook: Dayong Du: Gateway.

[4] Gates, Alan, and Daniel Dai. Programing Pig: Dataflow Scripting with Hadoop. 2 edition. O'Reilly Media, 2016.

[5] Urmila, R. 2016. "Big Data Analysis: Comparision of Hadoop MapReduce, Pig and Hive Dr. Urmila R. Pol Assistant Professor, Department of Computer Science, Shivaji University, Kolhapur, India" Vol. 5, Issue 6, June 2016 Copyright to IJIRSET.

[6] Blokdyk, Gerardus. MapReduce Complete Self-Assessment Guide. CreateSpace Independent Publishing Platform, 2017.

[7] Erraissi, A., And Belangour, A. (2018). Data sources and ingestion big data layers: meta-modeling of key concepts and features. International Journal of Engineering and Technology, 7(4), 3607-3612.

[8] Erraissi A., Belangour A. (2019) Capturing Hadoop Storage Big Data Layer Meta-Concepts. In: Ezziyyani M. (eds) Advanced Intelligent Systems for Sustainable Development (AI2SD'2018). AI2SD 2018. Advances in Intelligent Systems and Computing, Flight 915. Springer, Ham

[9] Erraissi Allae, and Abdessamad Belangour. "Hadoop Storage Big Data Layer: Meta-Modeling of Key Concepts and Features." International Journal of Advanced Trends in Computer Science and Engineering 8, No. 3 (2019): 646-53.

[10] Erraissi Allae, and Abdessamad Belangour. "Meta-Modeling of Big Data visualization layer using On-Line Analytical Processing (OLAP)." International Journal of Advanced Trends in Computer Science and Engineering 8, No. 4 (2019).

[11] Erraissi, Allae, and Abdessamad Belangour. "An Approach Based On Model Driven Engineering For Big Data Visualization In Different Visual Modes." International Journal of Scientific & Technology Research (2020).

[12] Erraissi Allae, and Abdessamad Belangour. "A Big Data Security Layer Meta-Model Proposal." Advances in Science, Technology and Engineering Systems Journal 4, No. 5 (2019). https://doi.org/10.25046/aj040553..

[13] Erraissi, Allae, and Abdessamad Belangour. Meta-Modeling of Big Data Management Layer. International Journal of Emerging Trends in Engineering Research 7, 7, 36-43, 2019. https://doi.org/10.30534/ijeter/2019/01772019..

[14] Royer, Jean-Claude, and Hugo Arboleda. Model-Driven and Software Product Line Engineering. 1st Edition. London, UK: Hoboken, NJ, USA: Wiley-ISTE, 2012.

[15] Ting, Kathleen, and Jarek Jarcec Cecho. Apache Sqoop Cookbook: Unlocking Hadoop for Your Relational Database. 1 edition. Sebastopol, CA: O'Reilly Media, 2013.

[16] Zaharia (M.), Borthakur (D.), Sarma (J. S.), Elmeleegy (K.), Shenker (S.) and Stoica (I.). Job Scheduling for Multi-User MapReduce Clusters. Technical Report n UCB/EECS-2009-55, EECS Department, University of California, Berkeley, April 2009.

[17] Jin (J.), Luo (J.), Song (A.), Dong (F.) and Xiong (R.). BAR: An Efficient Data Locality Driven Task Scheduling Algorithm for Cloud Computing. In: Proc. of the 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid). 295-304. Newport Beach, CA, May 2011.

[18] Ibrahim (S.), Jin (H.), Lu (L.), Wu (S.), He (B.) and Qi (L.). LEEN: Locality/Fairness-Aware Key Partitioning for MapReduce in the Cloud. In: Proc. Of the Second IEEE International Conference on Cloud Computing Technology and Science (CloudCom). 17-24. Indianapolis, IN, November 2010.

[19] Seo (S.), Jang (I.), Woo (K.), Kim (I.), Kim (J.-S.) and Maeng (S.). HPMR: Prefetching and Pre-shuffling in Shared MapReduce Computation Environment. In: Proc. IEEE International Conference on Cluster Computing (Cluster). New Orleans, LA, September 2009.

[20] Su (Y.-L.), Chen (P.C.), Chang (J.B.) and Shieh (C.-K.). Variable-Sized Map and Locality-Aware Reduce on Public-Resource Grids. FGCS, 27, n6, June 2011, 843-849.

[21] Veeravalli (B.), Ghose (D.), Mani (V.) and Robertazzi (T.). Scheduling Divisible Loads in Parallel and Distributed Systems. IEEE Computer Society Press, 1996, 292p.

[22] Berlinska (J.) and Drozdowski (Mr.). Scheduling Divisible MapReduce Computations. Journal of Parallel and Distributed Computing, 71, n3, March 2010, 450-459.

[23] Dean, Jeffrey, and Sanjay Ghemawat. "MapReduce: a flexible data processing tool." Communications of the ACM 53.1 (2010): 72-77.

[24] Condie, Tyson, et al. "MapReduce online." Nsdi. Vol. 10. No. 4. 2010.

[25] Dean, Jeffrey, and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters." Communications of the ACM 51.1 (2008): 107-113.

[26] Tian, Chao, et al. "A dynamic mapreduce scheduler for heterogeneous workloads." 2009 Eighth International Conference on Grid and Cooperative Computing. IEEE, 2009.

[27] Lam, Chuck. Hadoop in action. Manning Publications Co., 2010.

[28] Olston, Christopher, et al. "Pig latin: a not-so-foreign language for data processing." Proceedings of the 2008 ACM SIGMOD international conference on Management of data. 2008.

[29] Melton, Jim, and Alan R. Simon. Understanding the new SQL: a complete guide. Morgan Kaufmann, 1993.

[30] Vora, Mehul Nalin. "Hadoop-HBase for large-scale data." Proceedings of 2011 International Conference on Computer Science and Network Technology. Vol. 1. IEEE, 2011.

[31] Banane, Mouad, and Abdessamad Belangour. "A new system for massive RDF data management using Big Data query languages Pig, Hive, and Spark." International Journal of Computing and Digital Systems 9.2 (2020): 259-270.

[32] Geng, Yifeng, et al. "SciHive: Array-based query processing with HiveQL." 2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications. IEEE, 2013.

[33] Geng, Yifeng, et al. "SciHive: Array-based query processing with HiveQL." 2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications. IEEE, 2013.

[34] Borthakur, Dhruba. "HDFS architecture guide." Hadoop Apache Project 53.1-13 (2008): 2.

[35] Jouault, Frédéric, et al. "ATL: A model transformation tool." Science of computer programming 72.1-2 (2008): 31-39.

[36] Kalna, Fatima, et al. "A Scalable Business Intelligence Decision-Making System in the Era of Big Data." International Journal of Innovative Technology and Exploring Engineering (2019).

[37] Banane, Mouad, Allae Erraissi, and Abdessamad Belangour. "SPARQL2Hive: An approach to processing SPARQL queries on Hive based on meta-models." 2019 8th International Conference on Modeling Simulation and Applied Optimization (ICMSAO). IEEE, 2019.

[38] A. Erraissi, M. Banane, A. Belangour and M. Azzouazi, "Big Data Storage using Model Driven Engineering: From Big Data Meta-model to Cloudera PSM meta-model," 2020 International Conference on Data Analytics for Business and Industry: Way Towards a Sustainable Econamey (ICDABI), Sakheer, Bahrain, 2020, pp. 1-5, doi: 10.1109/ICDABI51230.2020.9325674.

[39] A. Erraissi and M. Banane, "Managing Big Data using Model Driven Engineering: From Big Data Meta-model to Cloudera PSM meta-model," 2020 International Conference on Decision Aid Sciences and Application (DASA), Sakheer, Bahrain, 2020, pp. 1235-1239, doi: 10.1109/DASA51403.2020.9317292.

[40] Erraissi, Allae, Banane Mouad, and Abdessamad Belangour. "A Big Data visualization layer meta-model proposition." 2019 8th International Conference on Modeling Simulation and Applied Optimization (ICMSAO). IEEE, 2019.

[41] Kleppe, Anneke G., et al. MDA explained: the model driven architecture: practice and promise. Addison-Wesley Professional, 2003.

Allae Erraissi is a PhD on Computer Science from Hassan II University, Faculty of Sciences Ben M'Sik, Casablanca, Morocco. His main interests engaged in research on various aspects of Information technologies namely Model-Driven Engineering approaches and their applications on new emerging technologies such as Big Data, Cloud Computing, Business Intelligence, Internet of Things, etc.