International Journal of Computing and Digital Systems
ISSN (2210-142X)
Int. J. Com. Dig. Sys. #, No.# (Mon-20..)

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

E-mail address: erraissi.allae@gmail.com

Received ## Mon. 20##, Revised ## Mon. 20##, Accepted ## Mon. 20##, Published ## Mon. 20##

Abstract: Big Data processing is done using MapReduce which is a clustered data processing framework. Composed of Map and Reduce functions, it distributes data processing tasks between different computers, then 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 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. Big Data 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]. Hadoop remains the main Big Data platform today. Based on Java, the open-source Hadoop framework is used 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 was originally a Facebook project that links the SQL world to Hadoop. It allows the execution of SQL queries on a Hadoop cluster in order to analyze and aggregate data. The SQL language is called HiveQL [3]. It is a visualization language only, which is why only "Select" instructions are supported for data manipulation. In some cases, developers must map between data structures and Hive. There is also Pig which is originally a Yahoo project which allows querying Hadoop data from a scripting language [4]. Unlike Hive, Pig is based on a high-level PigLatin language that allows you 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 systems. 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.

In this paper, we continue to apply techniques related to model engineering to standardize concepts at the Big Data level. In previous work [7] we proposed a meta-modeling of the layers: Data Sources and Ingestion. Then, we proposed a meta-modeling for the other layers of a Big Data system which are: Storage [8,9], Visualization [10], and Security [11]. This work is a progress report on our first proposal for a meta-modeling of the Big Data Management layer [12]. Our goal is to provide a generic Big Data system based on model engineering [13] to address the problem of a large number of Big Data solutions.

2. RELATED WORK

As part of our research project, we continue in this paper the application of techniques related to model engineering to standardize concepts at the Big Data level. In previous work [7] we proposed a meta-modeling of the layers: Data Sources and Ingestion. Then, we proposed a

E-mail: erraissi.allae@gmail.com

meta-modeling for the other layers of a Big Data system which are: Storage, Visualization, and Security. This work is a progress report on our first proposal for a meta-modeling of the Big Data Management layer [12]. The main paradigm used to process large amounts of data is the MapReduce [6], plus other complementary tools like Pig [4], Hive [3], Sqoop [14], etc. We treat in this paper 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 written by these two languages transform to MapReduce jobs according to the classic Big Data architecture. Our goal is to consolidate a generic Big Data system based on model 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 [15] improves this locality by introducing a delay before migrating a task to another node, if the preferred node is not available. The BAR [16] algorithm aims to approach the optimal data distribution taking into account an initial configuration that will be dynamically adapted.

LEEN [17] is an algorithm for partitioning intermediate keys that aim to balance the duration of reduction while trying to reduce bandwidth consumption during shuffle. This algorithm is based on statistics of the frequency of the appearance of intermediate keys in an attempt to create balanced partitions and optimize data transfers. The HMPR algorithm [18] proposes a pre-shuffling which tends to reduce the quantity of data to be transferred as well as the number of transfers. For this, 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 runtime environment [19], targeting the grids, adapts the amount of data to be processed 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 application can be considered as a set of divisible tasks since the data to be processed can be distributed equally between the map instances. It is therefore possible to apply results from the theory of divisible tasks [20] to this type of application. This is the approach that was followed by Berli´nska and Drozdowski [21]. In this article, the authors consider an execution environment in which the number of compute nodes is greater than the number of communications that can take place simultaneously without causing contention. To avoid the appearance of this phenomenon, they propose to model the execution of a MapReduce application 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 certain 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 Deanetal. In a MapReduce program, two types of operations are chained to perform a calculation: the Map operation and the Reduce operation. All of 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. More precisely, let and be three sets of keys and three sets of values. 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:

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

These operations are designed to allow easy distribution of the calculations. On a computing cluster, each machine processes only part of the data, so as 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. 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.

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. 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.

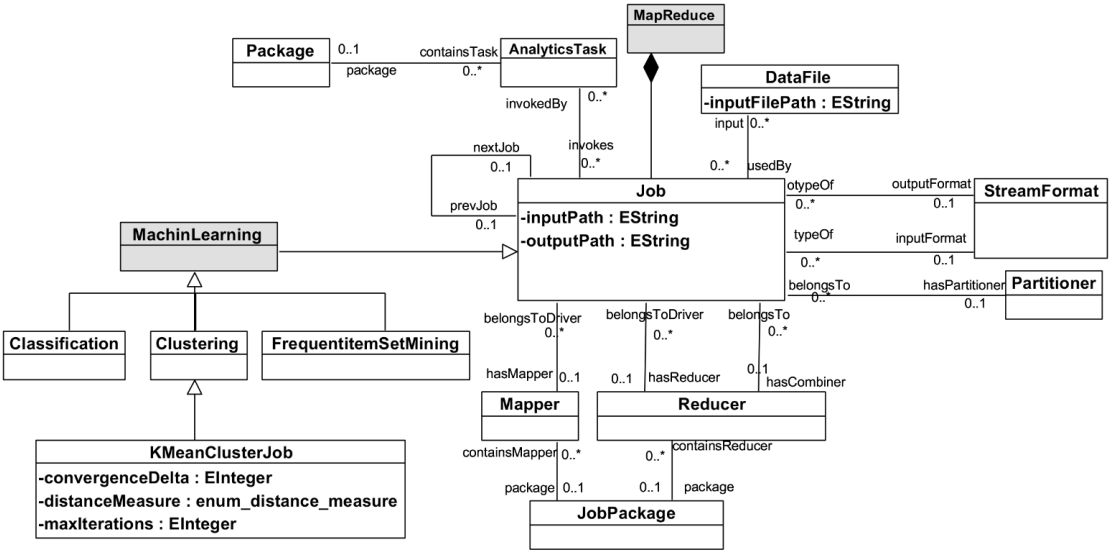

Figure 1. MapReduce meta-model proposed.

4. PIG

Apache Pig is a software created by Yahoo. It allows us to write useful treatments on data, without undergoing the complexity of Java. The goal is to make Hadoop accessible to non-computer scientists: physicists, statisticians, mathematicians, etc. Pig offers a scripting language called "Pig Latin". This language is called "Data Flow Language". Its instructions describe processing on a stream of data. Conceptually, it looks like a Unix tube; each command changes the flow of data passing through it. Pig Latin also makes it possible to build much more varied and non-linear treatments. Pig translates Pig Latin programs into MapReduce jobs and integrates the results into the flow.

A. *Example of a Pig program*

This program displays the 10 youngest adults extracted from a csv file containing 3 columns: identifier, name, and age.

```
personnes = LOAD 'personnes.csv' USING PigStorage(';')
        AS (userid:int, nom:chararray, age:int);
jeunesadultes = FILTER personnes BY age >= 18 AND age < 24;
classement = ORDER jeunesadultes BY age;
resultat = LIMIT classement 10;
DUMP resultat;
```

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 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 SGBDs.

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.

```
nom1 = LOAD ... ;
nom2 = FILTER nom1 ... ;
nom3 = ORDER nom2 ... ;
nom4 = LIMIT nom3 ... ;
```

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.

```
nom4 = 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:

```
nom1 = LOAD ... ;
nom2 = FILTER nom1 ... ;
nom3 = FILTER nom1 ... ;
nom4 = JOIN nom2 ..., nom3 ;
```

5) Relationships and types

A relation is an ordered collection of tuples which 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, nom:chararray, prenom:chararray, photo:bytearray,
    ancienneté:int, salaire: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 (
    nom:chararray,
    P1:tuple(x1:int, y1:int, z1:int),
    P2:tuple(x2:int, y2:int, z2:int));
DUMP segments;
```

8) Complex schema (bags)

You 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:

```
(nom:chararray, Points:{tuple(x:int, y:int, z: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 is the contents of the heros.csv file:

```
[nom#asterix,metier#guerrier]
[nom#tintin,metier#journaliste]
[nom#spirou,metier#groom]
```

It is made a relationship by:

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

Note: All these constructions, tuple, map and bags 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 the meta-model that we proposed for PIG:

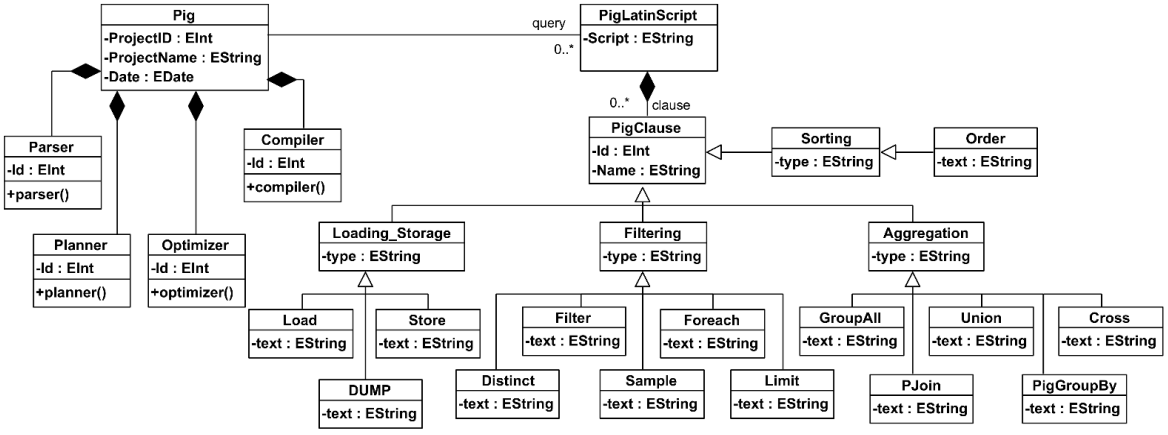

Figure 2. Meta-model of PIG.

5. HIVE

Hive simplifies work with a database like HBase or CSV files. Hive allows you to write queries in a language inspired by SQL and called HiveQL. These requests are turned into MapReduce jobs. To work, simply define a diagram that is associated with the data. This diagram gives the names and types of columns and structures the information into tables that Can be used by HiveQL.

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 releves (
        idreleve STRING,
        annee INT, ...
        temperature 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. Types HiveQL

Hive defines the following types:

- 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 expressed in number of seconds. Nanoseconds since 01/01/1970 UTC
- structured data as with Pig:
  - ARRAY indicates that there is a list of type
  - 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 nom (schéma) ROW FORMAT DELIMITED descr du 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's on HDFS in the table:

```
LOAD DATA INPATH '/share/noaa/data/186293'
    OVERWRITE INTO TABLE releves;
```

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 SGBD, 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, JOIN, WHERE, GROUP BY, HAVING, ORDER BY, LIMIT.

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/meteo/chaud'
SELECT annee,mois,jour,temperature
FROM releves
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 the meta-model we proposed for Hive and its HiveQL query language:

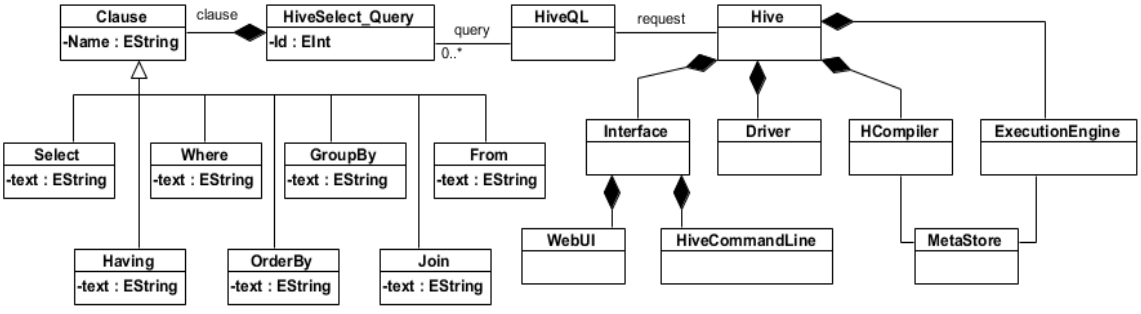

Figure 3. Hive meta-model.

6. TRANSFORMATION

After defining the meta-models of MapReduce, Pig, and Hive. In this section, we present the transformation rules used to pass from generic meta-models of PIG and Hive query languages to the 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 all the transformations, we chose the ATL transformation language. We now present extracts from the ATL code that we used to transform the meta-models proposed for Pig and Hive to the meta-model proposed for the MapReduce. These defined meta-models present the PIM (Platform Independent Model) level according to the architecture led by the 'MDA' models.

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. All these tools were installed on a machine with a 2.70 GHz Intel (R) Core (TM) i7 processor. With a storage space of 2 TB and a RAM memory 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:

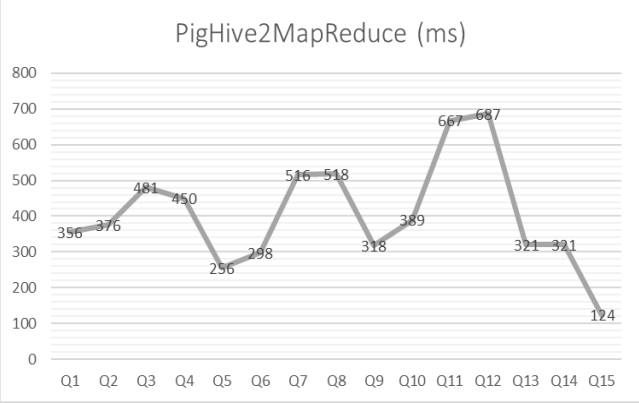

Figure 5. Transformation time of Pig requests on dataset1.

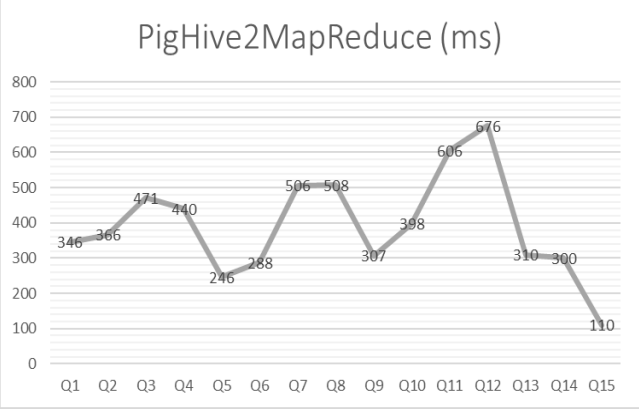

Figure 6. Transformation time of Pig requests on dataset2.

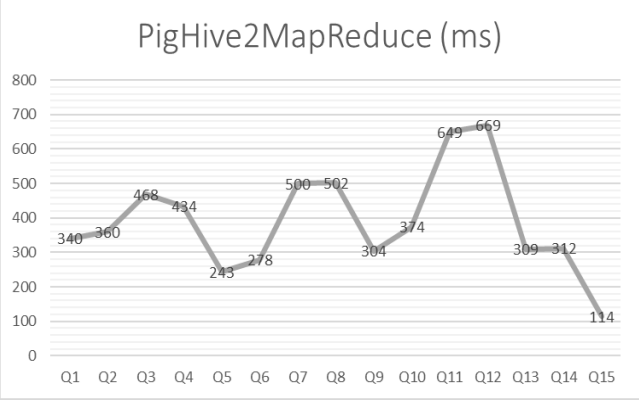

Figure 7. Transformation time of Pig requests on dataset3.

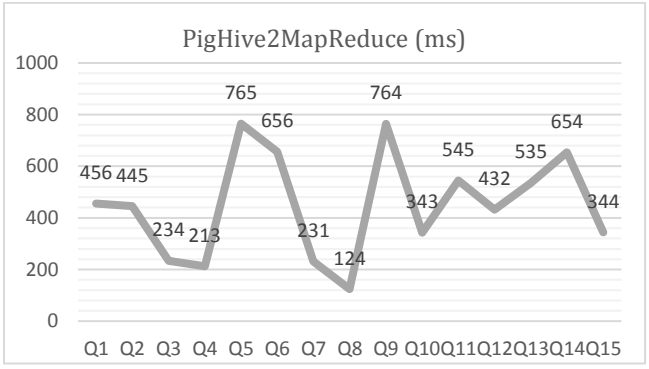

Figure 8. Transformation time of Hive requests on dataset1.

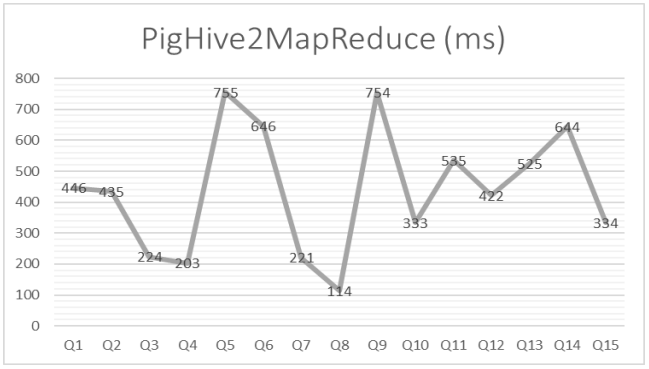

Figure 9. Transformation time of Hive requests on dataset2.

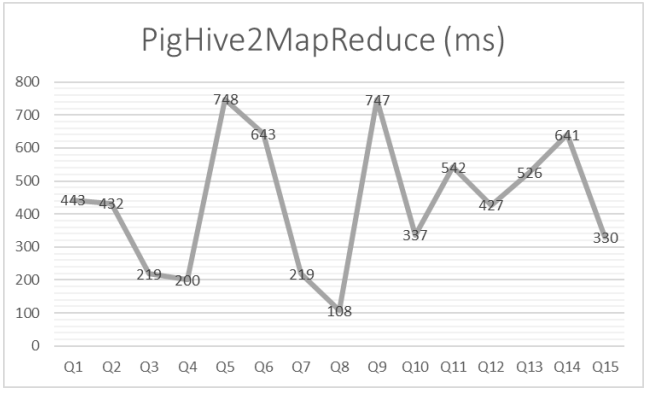

Figure 10. Transformation time of Hive requests on dataset3.

The results obtained from applying the transformations using the ATL transformation language show that the time to transform Pig Latin or HiveQL queries is very fast. 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. In continuous efforts, this work complements the meta-model already proposed for the management layer. Exactly 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

Given a large number of Big Data solutions available today in the market. We have noted the diversity of solutions and the non-interoperability between them. So, the application of techniques related to model engineering will standardize Big Data concepts. In our work, we aim to propose a universal meta-modeling of a Big Data system. Our proposals will be considered as a standard for Big Data.

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. "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..

[12] 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..

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

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

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

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

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

Allae Erraissi Allae Erraissi is a Ph.D. on computer science at the Faculty of Sciences Ben M'Sik at the Hassan II University, Casablanca, Morocco. He won his master's degree in information sciences and Engineering from the same University in 2016 and is currently working as a Mathematics teacher in a High school in Casablanca, Morocco. His main interests are the new technologies namely Model-driven engineering, Cloud Computing, and Big Data.