Modeling Apache Hive based applications in Big Data architectures

Enrico Barbierato
DI, Università degli Studi di Torino
corso Svizzera, 185
Torino, Italy
enrico.barbierato@mfn.unipmn.it

Marco Gribaudo
DEIB, Politecnico di Milano
via Ponzio, 34/5
Milano, Italy
gribaudo@elet.polimi.it

Mauro Iacono
DSP, Seconda Università degli Studi di Napoli
viale Ellittico, 31
Caserta, Italy
mauro.iacono@unina2.it

ABSTRACT

Performance prediction for Big Data applications is a powerful tool supporting designers and administrators in achieving a better exploitation of their computing resources. Big Data architectures are complex, continuously evolving and adaptive, thus a rapid design and verification modeling approach can be fit to the needs. As a result, a minimal semantic gap between models and applications would enable a wider number of designers to directly benefit from the results. The paper presents a multiformalism modeling approach based on a one-to-one mapping of Apache Hive querying primitives to modeling primitives. This approach exploits a combination of proper Big Data specific submodels and Petri nets to enable modeling of conventional application logic.

1. INTRODUCTION

The Big Data area is a recently emerged application field, resulting from the technological advancement and the increasing requirements coming from the need for processing enormous, continuously increasing amounts of low cost and low quality data. Commercial and scientific applications exploit massive data processing to produce new knowledge from wide numbers of sensors or to support complex information systems with thousands or millions of users or items, such as modern social networks (e.g. Facebook) or web based comprehensive services (e.g. Google).

Such enormous and evolving amounts of data call for very complex storage and computing resources, including thousand of components organized in computing nodes. Typical computations in Big Data are constituted by massively parallel executions of relatively simple elaborations. In order to manage with complexity, a specific development paradigm emerged, namely Map-Reduce, which rapidly spread and was implemented by different vendors or providers. Being such paradigm very elementary with respect to the needs of evolved applications, more abstract tools have been developed as well, offering more abstract primitives, such as high level query languages (similar to e.g. SQL).

The introduction of such abstractions is a big relief for developers, but can complicate the life of designers when taking decisions based on performance evaluation, as it additionally complicates the architecture. This paper proposes a modeling technique supporting designers in evaluating performances of high level Big Data applications. The originality of this approach consists of the possibility of modeling i) conventional aspects of the applications by means of Petri nets and/or queuing networks (therefore using multiformalism modeling techniques) and ii) Big Data related aspects of the application by means of a specific modeling language that mimics querying primitives.

The paper is organized as follows: Section 2 resumes related works, Section 3 describes the modeling approach, Section 4 presents an example, Section 5 gives an in-depth analysis of the solution process, Section 6 reports about results of example evaluation, Section 7 draws conclusions.

2. RELATED WORKS

2.1 A primer on Big Data and Apache Hive

Big Data is an application field that focuses on exploiting very large and non-structured databases. Open issues are i) scalability of computing and data storage architectures, ii) querying and processing technologies, iii) planning techniques and iv) fault tolerance (see [35, 29, 14, 23, 15, 26]). The main available solutions are based on Apache Hadoop ([3, 34]), which seem to be the main reference, Microsoft Dryad ([1, 27]), or Oozie ([31]), while other proposals are based on NoSQL databases such as MongoDB ([30]) and Apache Cassandra ([2]).

The reference programming paradigm is map-reduce, introduced by Google. This paradigm supports massively distributed computations on data organized in shards (NOSQL structures over data nodes). A shard contains table-like structures, that only store a given number of rows of a complete table, to balance the workload over the system. Computing is performed in stages consisting of a map phase and a reduce phase, in a pipeline. A map phase triggers a run of the desired task on all involved shards, while a reduce phase collects and processes the outputs.

Higher level tools for Big Data have been produced, generally on top of an implementation of the map-reduce paradigm and NoSQL databases, to enable the development of complex applications. This paper considers the case of Apache Hive [4, 16]. Apache Hive is an infrastructure designed to

VALUETOOLS 2013, December 10-12, Torino, Italy

DOI 10.4108/icst.valuetools.2013.254398

build data warehouse applications on top of Apache Hadoop, capable of executing data summarization, queries, and database analysis, by means of a high level abstraction encapsulating such functionalities in a sort of SQL-like query language (Hive Query Language, HQL in brief). This approach lowers the cultural gap for programmers and simplifies the migration and the scaling-up of existing SQL-based data warehousing systems to Big Data architectures. Hive does not rely on relational-like tables, nor allows record-level operations, as HQL queries are translated into map-reduce pipelines executed in batch. Consequently, Hive does not operate in real time, as map-reduce jobs have a non negligible setup and cleanup time. A proper use of Hive in Big Data systems implies that query processing time is significantly bigger than setup and cleanup time, making the former rather not influent on the overall Hive operation. Data on which Hive is meant to operate is organized in big chunks, namely partitioned tables, that are distributed into files, named after the chosen partitioned criteria, so that all data stored in records sharing the same partition key values assigned are located in the same file.

Performance evaluation of Big Data applications is still an open problem: in [21], the authors present a performance analysis approach fit for helping designers and administrators in fine-tuning Big Data cloud environments by a posteriori analysis. A map-reduce applications workload generator is used in [18] to evaluate performance trade-offs; in [32] cloud-based data management systems benchmarking is analyzed. In [17] a modeling approach for performance prediction of adaptive Big Data architecture is studied. In [36], a performance evaluation framework is presented. This approach estimates the completion time of programs that are reused for processing a regularly incoming new data as a function of a new dataset and the cluster resources. Finally, [9] presents a modeling language to model low level aspects of Big Data applications, on which this work is based.

2.2 Multiformalism modeling

Multiformalism modeling is a research field that aims to support modelers by enabling them to exploit the most profitable aspects of different modeling formalisms together. The advantage of using a multiformalism approach is twofold: on the one hand, it allows the modeler to choose familiar formalisms to represent the problem; on the other hand, it provides the capability to describe rapidly the different aspects of the problem in the most natural and appropriate way. Möbius [19, 20], OsMoSys [33, 22] and SIMTHESys [24, 6] are multiformalism frameworks offering different perspectives and solution infrastructures. While Möbius is the most mature and efficient tool, OsMoSys provides a sophisticated extensible object-oriented approach model techniques and formalism specifications. Performance predictions are discussed in [13] by presenting Palladio Component Model, which specifies component-based software architectures in a parametric way. Finally, SIMTHESys presents mechanisms for the automatic generation of solvers for user-defined formalisms.

In this paper the SIMTHESys framework is used to support the development of a custom formalism mimicking the HQL language. SIMTHESys formalisms (used already in [9]) are specified in terms of formalism elements, in the form of nodes and arcs of a graph, characterized by properties (that qualify elements with related data) and behaviors (describing how they interact with other elements according to property values). This approach allows the automatic synthesis of custom solvers that implement behaviors by building layers over elementary solvers (solving engines) provided by the framework. Examples of formalisms that have been used for the analysis of case studies can be found in [6, 7, 10, 25, 24, 5, 11, 12, 9, 17].

3. MODELING APPROACH

Performances of Big Data applications depend i) on the application logic, ii) on the system architecture on which the application is executed in a given period of time and iii) on how data are partitioned over shards. Data distribution and architecture should adapt to the performance needs, that in turn depend on workloads and external solicitations. Modeling an application requires: i) a mean to capture the architecture at the needed level of detail; ii) a mean to correctly map the dependence of execution patterns over the architecture; iii) a mean to account for general application logic; and iv) a mean to describe the effects of the external environment over the application. Using a multiformalism approach, the four aspects can be potentially faced by different formalisms, specializing the description of the four contributions to system performances in different submodels that can be designed separately and reused. In SIMTHESys, this requires a bridging formalism[1], that is a formalism allowing the definition of models capable of combining submodels based on different formalisms.

Two different high level domain oriented formalisms are proposed to manage the application logic and the system architecture. The advantage of this choice consists of adopting a natural description for what is specifically related to the Big Data domain, as a lower level description for the same concepts would be further more complicated.

3.1 Modeling the architecture

This paper extends the work presented in [9], which provides an in-depth description of the SIMTHESys formalism SIMTHESysBigData. SIMTHESysBigData has been developed to describe at low application level the architecture of a Big Data system based on Apache Hadoop and simple map-reduce pipelines. The formalism offers the elements represented in Fig. 1 (that also indicates their main properties).

The architecture is described by means of the following structural elements: i) *Dataset*, representing a logical/physical group of data; ii) *Shards*, representing a group of shards on which a Dataset is stored and iii) *Temporary Dataset*, which represents temporary tables needed by map-reduce pipelines execution. Map-reduce pipelines and their inputs are represented by the following operational elements: i) *Trigger*, which is a Poisson-distributed arrivals data source generating input loads towards a Dataset; ii) *Map*, the map phase in a map-reduce task; iii) *Reduce*, the reduce phase in a map-reduce task and iv) *Local actions*, non Big Data related computations that are executed on a computing node. Structural and operational elements are connected by the follow-

[1]This is technically not mandatory, as elements from the different formalisms could be natively enabled to interact with elements from other formalisms: but it is advisable, to keep the formalisms independent from each other and improve their reusability, and separate formalism design from formalism interconnection design.

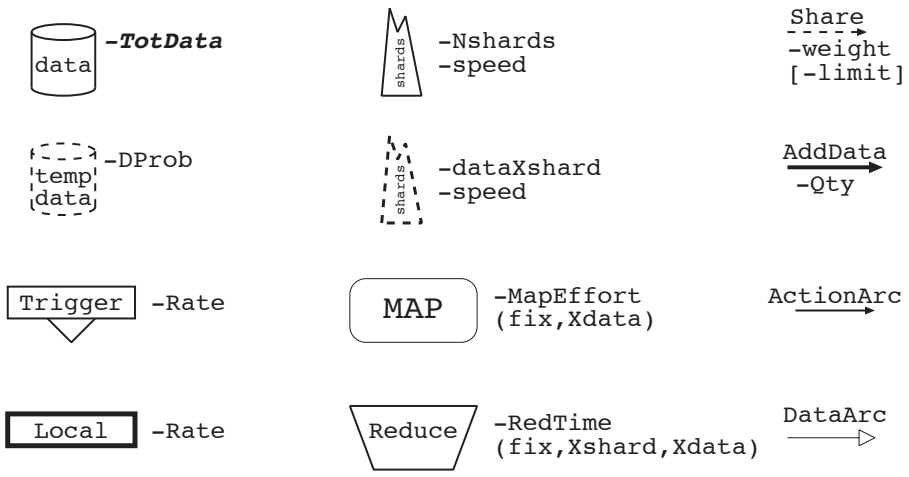

Figure 1: Elements of the SIMTHESysBigData and the SIMTHESysHQL formalisms

ing arc elements: i) *Share*, which binds a logical/physical group of data and its corresponding shard; ii) *AddData*, which connects a Trigger and the Dataset to which new data are stored; iii) *ActionArc*, which describes the execution path between operational elements in a map-reduce pipeline and finally iv)*DataArc*, indicating the Dataset on which a Map operates.

Properties and behaviors of the elements of the SIMTHESysBigData formalism are omitted for sake of space, but their description can be found in [9][2].

3.2 Modeling Apache Hive queries

With respect to the system architecture, this paper uses an approach that is similar to the one applied in [8]. The main idea is to allow the modeler to use HQL itself as modeling language, and exploit model transformation to mimic the execution of the query by Apache Hive. The formalism is composed of a single element, namely Query, which has a main property, containing the desired query.

This approach is viable since Apache Hive offers the EXPLAIN command. This command allows to understand how a query is internally managed by Hive, and provides the exact listing of the map-reduce jobs by which a query will be executed, given data distribution over shards. The command has been used in the model transformation that is applied in the solving process.

3.3 Modeling application logic and environment

For both application logic and the external environment, existing SIMTHESys Generalized Stochastic Petri Nets [28] (GSPN) formalism has been used. This choice is due to the generality and the popularity of this formalism, widely used to model performances related metrics of concurrent systems. The adoption of GSPN allows to describe the non Big Data part of the system with flexibility and independently by its actual architecture, that would not be significant with respect to the goals of this paper, while considering the general behavior of the represented subsystems. The SIMTHESys GSPN formalism is completely coherent with GSPN but adds the support for multiformalism integration.

[2]The complete SIMTHESys FDL description document for this formalism and the others can be obtained just sending an email to the authors, as well as the SIMTHESys MDL document for the example.

3.4 The bridge formalism

The bridge formalism is composed by three kinds of arc, that will here be designated after the formalisms used by the couple of submodels that they let interact.

The first kind is the GSPN to GSPN arc: its semantic is the same of a couple of GSPN arcs connecting a transition to a place and vice versa, also known as test arc. This arc connects a transition of a GSPN submodel to a place of a different GSPN submodel and enables the firing of the transition if the place is marked.

The second kind is the GSPN to SIMTHESysBigData arc: its semantic generates a request of operation towards a Map element if a transition fires. This arc connects a transition of a GSPN submodel to a Map of a SIMTHESysBigData submodel and acts on it as an ActionArc.

The third kind is the GSPN to SIMTHESysHQL arc: its semantic triggers the execution of a SIMTHESysHQL submodel if a transition fires. This arc connects a transition of a GSPN submodel to a SIMTHESysBigData submodel.

From a graphical point of view, given the impossibility of incidental confusion due to the fixed and unambiguous nature of starting and ending point, all these arcs have been represented with the same symbol in the example model of Fig. 2.

4. AN EXAMPLE

The example considered describes a massive textual document data base, serving a number of users requesting search operations and is used to perform Big Data queries. Users are typically working in peak hours, but the system is somehow used by some users also in off-peak hours. Big Data operations are executed on the system when it is not in peak hours. Big Data operations are decided and launched by a controller unit.

The model of the system is in Fig. 2 and is based on a multiformalism approach.

The behavior of users is described by a GSPN submodel (denoted by Users in Fig. 2). The peak hours situation is

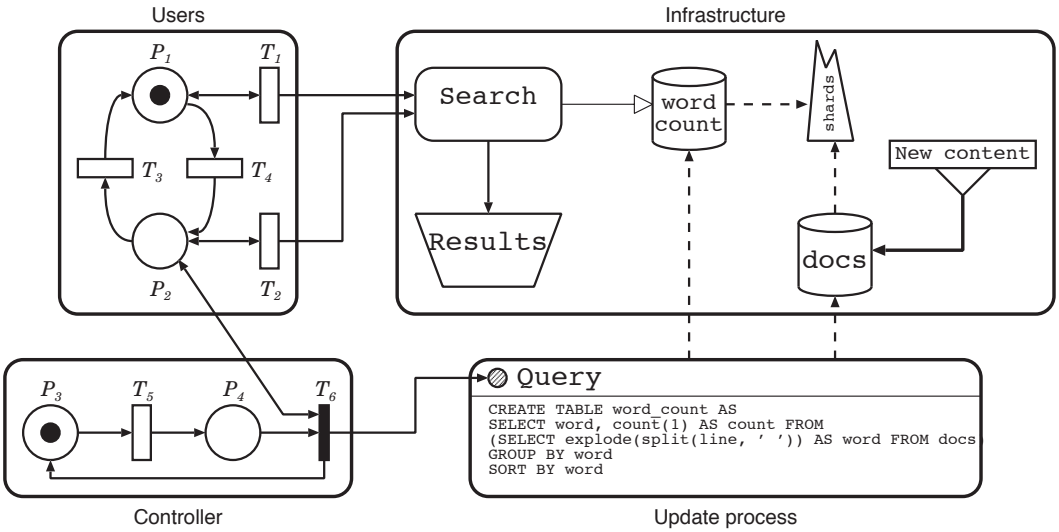

Figure 2: Example model

modeled by place P1. When P1 is marked, T1 is enabled and requests are generated towards the Infrastructure submodel with a constant rate. Transition T4 determines the rate at which the system switches to off-peak mode, represented by place P2. Place P2 enables requests towards Infrastructure by means of transition T2 (that, consequently, has a smaller rate with respect to T1), and authorizes the Controller submodel if it attempts a Big Data query execution. The system switches back to peak mode by transition T3. The ratio between the rates of transition T4 and transition T3 defines the relative duration of peak and off-peak periods.

The behavior of the controller is described by a GSPN submodel (denoted by Controller in Fig. 2). Place P3 defines the wait state of the controller, and is thus marked when the controller is not requesting a Big Data operation. T5 defines the rate with which a Big Data request should be attempted. When it fires, place P4 is marked, defining that the controller is ready to issue the request. Transition T6 activates a request by activating the Update process submodel, if the controller is ready for the request and the users submodel signals that the system is in off-peak mode: if the system is in peak mode, the request will be executed as soon as the system switches to off-peak mode.

The architecture of the data center is described by a SIMTHESysBigData submodel (denoted by Infrastructure in figure). The system is configured to execute an operation represented by map Search (that operates on the dataset Word Count, relying on shard Shards) and by reduce Results. Besides dataset Word Count, also dataset Docs relies on shard Shards. Dataset Docs is also updated by additional contents by trigger New content.

The query insisting on the Infrastructure submodel is described by a SIMTHESysHQL submodel (denoted by Query in figure). The example taken in account focuses on one of the Hive queries commonly used as a benchmarking application. The considered data base includes a table, named *Docs*, containing a set of lines extracted from a corpus of texts. The records of the table are composed of a single string field called *line*, containing one line of the considered documents. The query ([16]) used to perform the word count is the following:

```
CREATE TABLE word_counts AS
SELECT word, count(1) AS count FROM
(SELECT explode(split(line, ' ')) AS word FROM docs)
GROUP BY word
SORT BY word
```

5. SOLVING THE MODELS

The solution strategy for the model is composed of two main steps. The first step concerns the transformation of the SIMTHESysHQL submodel into the equivalent SIMTHESysBigData submodel, the second regards the solution of the resulting overall model by the proper solver. The resulting strategy mixes vertical multiformalism in the first step and horizontal multiformalism in the second step.

The solution workflow is detailed in Fig. 3. The user-provided model describing the system is processed by the Model Analyzer, in charge of analyzing the structure of the model to deliver the query SIMTHESysHQL submodel to the Transformation Engine and the rest of the model to the Model Generator.

The Transformation Engine is in charge of enacting the transformation of the first step. This is done by exploiting a properly configured Apache Hive installation (in a virtual machine or on a production site) returning the query execution plan for the desired HQL query by using the EXPLAIN command. The query execution plan is analyzed and translated in the corresponding query SIMTHESysBigData equivalent submodel.

The query SIMTHESysBigData submodel is then processed together with the rest of the model by the Model Generator, which rebuilds the overall model and generates the right arcs to bind properly the query SIMTHESysBigData submodel to the architecture SIMTHESysBigData submodel.

The resulting model is sent to the ad hoc multiformalism solver generated by SIMTHESys*ER*, the framework solver generation tool, by building together the behavioral descriptions of the used formalisms and the solving engines of the framework. The ad hoc solver can directly evaluate, with

analytical or simulative techniques, the model, and provides the desired results.

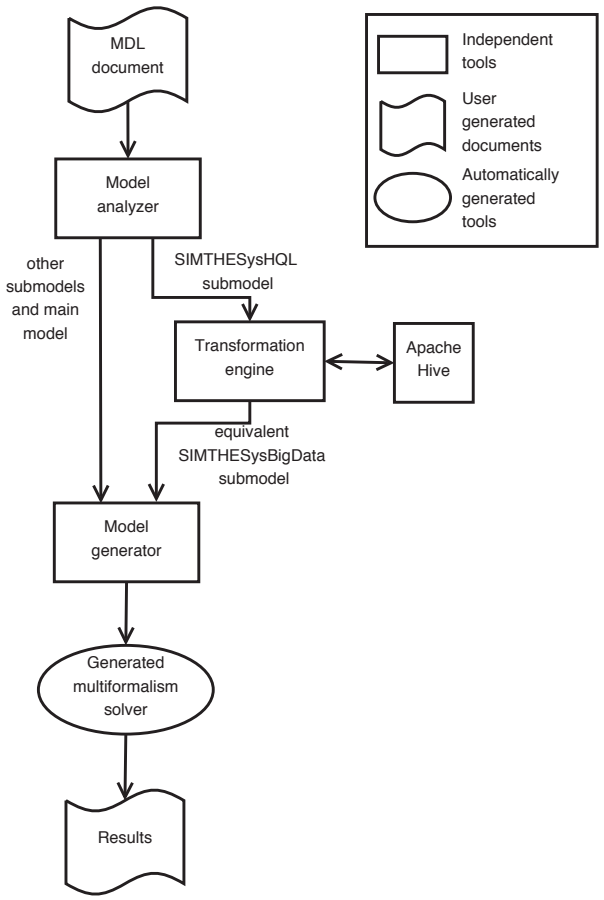

Figure 3: The solution workflow

5.1 The Transformation Engine

While details about the second step, the involved tools and the generalities about SIMTHESys can be found in the papers cited in Section 2, the Transformation Engine and the rest of the concepts presented while describing the solution workflow in Fig. 3 are new additions to the SIMTHESys framework. A Transformation Engine can be regarded as a specialized solver that does not perform a quantitative evaluation, but is rather capable of analyzing a (sub)model written in a formalism and generating another (sub)model, written in another formalism, according to given transformation rules. A Transformation Engine can map between them elements from the source and target formalism or can perform a more complex model generation, eventually exploiting external tools, as in the case presented by this paper. From this point of view, the Model Generator in Fig. 3 is a Transformation Engine generating elements of the model using the bridge formalism.

Transformation from SIMTHESysHQL to SIMTHESysBigData is not a complex task. Map-reduce stages in a query, as resulting from an EXPLAIN command output, are converted into a sequence of a Map element and a Reduce element connected by an Action arc, while non map-reduce stages are directly represented as Local elements. For each stage the EXPLAIN output provides both input and output (either temporary or permanent) tables, which can be represented by Dataset or Temporary Dataset elements. Binding between the input tables and a map-reduce sequence can be represented by Data arc elements, while the production of new data into output tables by a map-reduce sequence can be represented by *AddData* arc elements. Finally, the event of query activation is represented by a Trigger element, connected to every Map element of a map-reduce stage that does not depend on other map-reduce stages by an ActionArc arc element. The corresponding Reduce elements is connected to the subsequent Map elements by another ActionArc arc element.

The Transformation Engine used in this paper adopts this logic and is based on the transformation algorithm in Fig. 4. Note that though algorithms are used in this paper for the sake of clarity, the discussions of their (essentially polynomial) complexity is out of scope.

```
FUNCTION HQLtoBigData(String $Query) : BigDataModel
BEGIN
  $model = NEW BigDataModel()
  $expQ = ExecuteQuery('EXPLAIN EXTENDED $Query')
  TranslateQuery($model, $expQ)
  ConnectModel($model, $expQ)
  RETURN $model
END
```

Figure 4: The SIMTHESysHQL to SIMTHESysBigData Model transformation algorithm

At line 3, `NEW BigDataModel()` is used to generate a new empty SIMTHESysBigData model. At line 4 the external Apache Hive installation is called to execute the *EXPLAIN* command on the target query, extracted from the query SIMTHESysHQL submodel. Lines 5 and 6 respectively invoke `TranslateQuery()` (Fig. 5) and `ConnectModel()` (Fig. 6) to generate the equivalent SIMTHESysBigData operational elements and to build them up into the submodel.

The elements generation algorithm is presented in Fig. 5. The `for each` at line 3 performs the generation of all parts in the query that are constituted by map-reduce jobs, while the `for each` at line 23 takes care of all other parts. At line 4 and 5, `$model.add($etype, $id)` adds a `$etype` element to the model (respectively a Map and a Reduce); at line 6 `$model.addArc($atype, $from, $to)` adds an Action arc that connects them. The `for each` at line 8 iterates over all input and output tables used by the query to generate Temporary Dataset and Dataset elements used by the Map and the Reduce. Finally, at line 16 the `if then else` creates the required Data and AddData arcs. The second `for each` adds Local elements (line 24) representing non map-reduce computations in the query. It merges also temporary and permanent tables in the model when required by a temporary table to permanent table move instruction, replacing the former with the latter (line 25).

Fig. 6 presents the algorithm that builds up the model fragments obtained by the previous algorithm into the query SIMTHESysBigData submodel. Line 2 adds to the submodel a Trigger element representing the query call; the `for each` statements at line 3 and 6 produce the required Action arcs, according to the execution order of the query stages obtained by EXPLAIN.

```
PROCEDURE TranslateQuery(BigDataModel $model, Stages $expQ)
BEGIN
  FOR EACH Map-Reduce_Stage $s IN $expQ
    $model.add('Map', 'map_$s')
    $model.add('Reduce', 'reduce_$s')
    $model.addArc('Action', 'map_$s', 'reduce_$s')
    $Tables = $s.getInTables() + $s.getOutTables()
    FOR EACH Table $t IN $Tables
      IF $t NOT IN $model THEN
        IF $t.isTemporary() THEN
          $model.add('Temporary Dataset', $t)
        ELSE
          $model.add('Dataset', $t)
        END IF
      END IF
      IF $t.isInput() THEN
        $model.addArc('Data', 'map_$s', $t)
      ELSE
        $model.addArc('AddData', 'reduce_$s', $t)
      END IF
    END FOR
  END FOR
  FOR EACH Non_Map-Reduce_Stage $s IN $expQ
    $model.add('Local', 'local_$s')
    IF $s.getType() = 'Move table $t1 in $t2'
       AND $t1.isTemporary() AND NOT $t2.isTemporary
       $model.MergeToPermanentTable($t1, $t2)
    END IF
  END FOR
END
```

Figure 5: The SIMTHESysHQL to SIMTHESysBigData Model transformation algorithm: elements generation

```
PROCEDURE ConnectModel(BigDataModel $model, Stages $expQ)
  $model.add('Trigger', 'Query')
  FOR EACH root_Stage $s of type $t IN $expQ
    $model.addArc('Action', 'Query', '$t_$s')
  END FOR
  FOR EACH Non_root_Stage $s of type $t IN $expQ
    FOR EACH Stage $d (type $td$ on which $s depends
      $model.addArc('Action', '$td_$d', '$t_$s')
    END FOR
  END FOR
BEGIN
END
```

Figure 6: The SIMTHESysHQL to SIMTHESysBigData Model transformation algorithm: connect stages

5.2 Transformation example

As for the output of the EXPLAIN command on the query from the example, the performed word count works as follows: as first, all the lines stored in the *docs* table are examined, and single words are extracted by the *split()* function. Each output is transformed into a row by the *explode()* function. Then, identical words are counted by the *count()* function and the *GROUP BY* statement. Results are then ordered by word (*ORDER BY* clause) and stored in the new word_counts table (*CREATE TABLE* statement). The execution plan is executed in five stages, from *Stage-0* to *Stage-4*, numbered by Apache Hive according to their lexicographical position in the query. The execution order is different, for the effects of data dependencies, and it is described as follows:

$$Stage\text{-}1 \rightarrow Stage\text{-}2 \rightarrow Stage\text{-}0 \rightarrow Stage\text{-}4 \rightarrow Stage\text{-}3$$

The first stage executes the most of the workload, via a map-reduce operation, in which the map phase extracts the words contained in the lines of the text stored in the DB, by executing both the *split()* and *explode()* commands. The reduce phase executes the grouping function producing the final word count. The output is stored in a temporary table, to be reused, as an un-sorted list of pairs (word, number of occurrences). The second stage performs sorting over the pairs, according to the word field, by using a map-reduce operation over the temporary table, in which the map phase does not perform any significant action and the reduce phase executes the actual sorting. The results are stored in a second temporary table. Finally, the third stage renames the temporary table as *word_count*, while the last two stages execute integrity operations (schema update to include the new table, and table statistics update to allow optimizations, respectively).

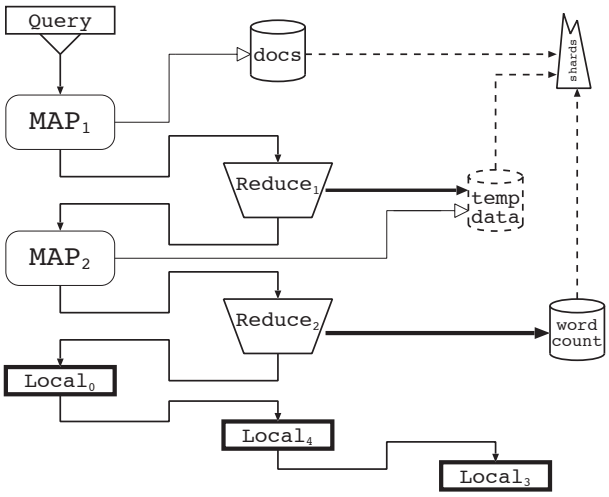

Figure 7: Generated SIMTHESysBigData equivalent for the query SIMTHESysHQL submodel of the example

Fig. 7 shows the resulting SIMTHESysBigData equivalent produced by the Transformation Engine. Every query actual execution stage has been translated to a *Map* element and *Reduce* element sequence fragment or to a *Local*

element. The transformation generated three tables (input, output and temporary tables used by the query), the second of which (*word_count*) has been represented as a permanent table (*Stage-0*, third in order of execution, executes a *Move* operation). Fig. 7 also shows the *Shards* element and the *Share* arcs generated by the Model Generator to map together the query SIMTHESysBigData submodel and the architecture SIMTHESysBigData submodel.

5.3 Model generator and second step of the solution process

The model generator replaces the query SIMTHESysHQL submodel with the query SIMTHESYSBigData equivalent, analyzes it and generates the overall SYMTHESysBigData submodel by producing *Share* arcs to connect properly *Dataset* elements and *Temporary Dataset* elements with the *Shards* modeling the physical infrastructure of the system. The newly obtained SIMTHESysBigData submodel replaces the processed one in the model.

The second step of the solution process is a SIMTHESys multiformalism model solution process and is performed by the SIMTHESys*ER* tool. The tool is used to automatically generate a custom solver that can handle the three residual formalism, obtained as in [9][3].

6. PERFORMANCE EVALUATION

We evaluate the model presented described in Section 4 with the parameters summarized in Table 1. To better suit the variability of the big-data environment, transitions have been chosen to be very fast: this allows the representation of a dynamic scenario with short epochs, containing a large number of events. The Map elements in the Map-reduce formalism, are parameterized by two parameters: a constant time, plus another parameter inversely proportional to the number of shards and directly proportional to the quantity of data on which the primitive is operating. This allows to model the speed-up that can be obtained thanks to the parallel operations of the map-reduce nodes composing the system. Reduce elements are instead characterized by three parameters: a constant time, a parameter proportional to the size of the dataset and inversely proportional to the number of shards, plus one factor proportional to the number of shards. The last parameter is used to capture the increased synchronization delay experienced in the reduce phase to collect results computed by the map phase. Query are also characterized by similar parameters, which are influenced by the speed of the DB, the number of shards over which the Hive is deployed, and the size of the considered dataset. For the sake of simplicity, only the aggregated mean service time of either the Map-reduce job, or of the entire query, are reported in Table 1. Specifically, sw refers to the switch from n refers to the number of shards and d to the size of the dataset. In this example, d is incremented by the firing of the $NewContent$ trigger, and reset after the execution of the query. We suppose that the query has to consider at least a document each time it is executed.

To show the features of the proposed technique, we look for the best number of shards that are required to minimize the response time for a given load. In particular, we scale both the firing rate in the high and low traffic states of the system (namely, T_1 and T_2, while T_3 and T_4 represents the firing rate in on and off peak switch respectively), and of the new content generation (trigger $NewContent$), of an identical rate $\alpha \in \{0.2, 0.4, 0.6, 0.8, 1\}$. Fig. 8 shows the perceived response time in logarithmic scale. Results were computed using discrete event simulation, for a simulation time of 50000s, with each measure computed by averaging 625 runs. Confidence interval were computed: although not shown for the sake of simplicity, the 95% confidence intervals have a variability that is less than 10% for all the considered performance indices.

[3]For further details about SIMTHESys*ER* the reader can refer to [24].

Table 1: default

T_1	$200{\cdot}\alpha$ req/s.	T_2	$20{\cdot}\alpha$ req/s.
T_3	1 switch/s.	T_4	2 switch/s.
T_5	50 req/s.	*New Content*	$10{\cdot}\alpha$ req/s.
Search	(aggregate) $0.5 + 0.02{\cdot}n + 4/n$ s.		
Query	(aggregate) $2 + 0.4{\cdot}n + 43.5{\cdot}d/n$ s.		

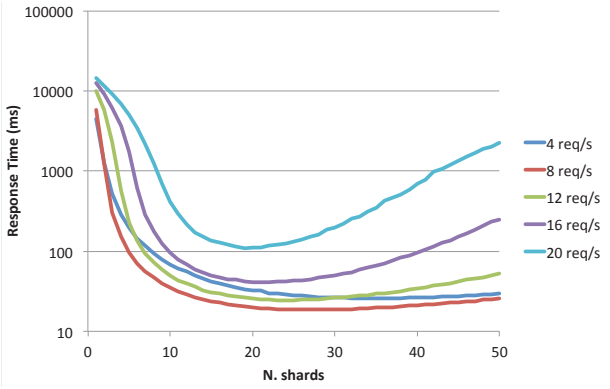

Figure 8: Response time as function of the number of shards, for different firing rates of transition T_2

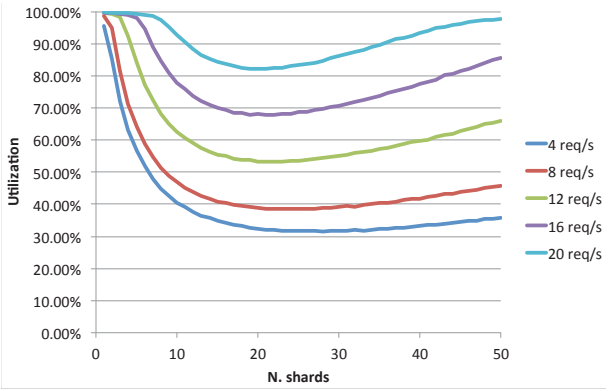

Figure 9: Utilization as function of the number of shards, for different firing rates of transition T_2

As we can see, the response time has a minimum: this happens because for a small number of shards, the lack of parallelization causes an high end-to-end delay. When however the number of shards becomes too big, the synchronization overhead becomes the dominant component of the process,

increasing the perceived response time. It is interesting to note that the response time for a load of $T_2 = 4$ req/s is greater than the one for $T_2 = 8$ req/s, and for $8 \leq n \leq 30$ it is also greater than the time measured with $T_2 = 12$ req/w. This comes from the fact that the indexing query is executed even if no new content has been generated since the last firing of T_5: this creates an extra overhead that is proportionally more visible in a scenario with a smaller load. Fig. 9, presents the utilization of the system. This clearly shows that the system saturates for a small number of shards for $T_2 \geq 8$ req/s. The finite response time in Fig. 8 is obtained only thanks to the finite simulation time imposed by the adopted solution method: this also justifies the flexions of the curves for small values of n for $T_2 \geq 12$ req/s. To summarize the results, if we have for example a load of $\alpha = 0.8$ (that is, $T_2 = 16$ req/s), the system cannot work with less than 8 shards. The optimal response time is instead obtained for $n \approx 23$ shards.

7. CONCLUSIONS

In this paper multiformalism techniques has been used on a horizontal and a vertical perspective: horizontal multiformalism has been used to compose the environment around the Big Data system and vertical multiformalism has been used to automatically obtain the model of the Big Data System by specifying it on two different abstraction levels. The approach hides the complexity of the system behind a structural, foundational description of the architecture and a logical description of the query operation, keeping the representation close to metaphors that are familiar for domain experts of the Big Data field.

Future works go towards an extension of the framework to explore other potentially useful vertical multiformalism application to other aspects (and other languages) of the Big Data domain, and the extension towards reliability and availability evaluation.

8. REFERENCES

[1] Microsoft Dryad, 2007.

[2] Apache Cassandra. Apache Cassandra web site, 2009.

[3] Apache Hadoop. Apache Hadoop web site, 2008.

[4] Apache Hive. Apache Hive web site, 2013.

[5] E. Barbierato, A. Bobbio, M. Gribaudo, and M. Iacono. Multiformalism to support software rejuvenation modeling. In *ISSRE Workshops*, pages 271–276. IEEE, 2012.

[6] E. Barbierato, M. Gribaudo, and M. Iacono. Defining Formalisms for Performance Evaluation With SIMTHESys. *Electr. Notes Theor. Comput. Sci.*, 275:37–51, 2011.

[7] E. Barbierato, M. Gribaudo, and M. Iacono. Exploiting multiformalism models for testing and performance evaluation in SIMTHESys. In *Proceedings of 5th International ICST Conference on Performance Evaluation Methodologies and Tools - VALUETOOLS 2011*, 2011.

[8] E. Barbierato, M. Gribaudo, and M. Iacono. Performance evaluation of nosql big-data applications using multi-formalism models. *Future Generation Computer Systems*, to appear, 2013.

[9] E. Barbierato, M. Gribaudo, and M. Iacono. A performance modeling language for big data architectures. In *Proceedings of High Performance Modelling and Simulation 2013, European Conference on Modelling and Simulation 2013, 27-30 May, 2013, Aalesund (Norway)*, 2013.

[10] E. Barbierato, M. Gribaudo, M. Iacono, and S. Marrone. Performability modeling of exceptions-aware systems in multiformalism tools. In K. Al-Begain, S. Balsamo, D. Fiems, and A. Marin, editors, *ASMTA*, volume 6751 of *Lecture Notes in Computer Science*, pages 257–272. Springer, 2011.

[11] E. Barbierato, M. Iacono, and S. Marrone. PerfBPEL: A graph-based approach for the performance analysis of BPEL SOA applications. In *VALUETOOLS*, pages 64–73. IEEE, 2012.

[12] E. Barbierato, G. D. Rossi, M. Gribaudo, M. Iacono, and A. Marin. Exploiting product form solution techniques in multiformalism modeling. *Electr. Notes Theor. Comput. Sci.*, to appear, 2012.

[13] S. Becker, H. Koziolek, and R. Reussner. Model-based performance prediction with the Palladio component model. In *Proceedings of the 6th international workshop on Software and performance*, WOSP '07, pages 54–65, New York, NY, USA, 2007. ACM.

[14] E. Bertino, P. Bernstein, D. Agrawal, S. Davidson, U. Dayal, M. Franklin, J. Gehrke, L. Haas, A. Halevy, J. Han, and Others. Challenges and Opportunities with Big Data. 2011.

[15] R. E. Bryant, R. H. Katz, and E. D. Lazowska. Big-data computing: Creating revolutionary breakthroughs in commerce, science, and society. In Computing Research Initiatives for the 21st Century. *Computing Research Association*, 2008.

[16] E. Capriolo, D. Wampler, and J. Rutherglen, editors. *Programming Hive - Data Warehouse and Query Language for Hadoop*. O'Reilly Media, 2012.

[17] A. Castiglione, M. Gribaudo, M. Iacono, and F. Palmieri. Exploiting mean field analysis to model performances of big data architectures. *Future Generation Computer Systems*, to appear, 2013.

[18] Y. Chen, A. S. Ganapathi, R. Griffith, and R. H. Katz. Towards Understanding Cloud Performance Tradeoffs Using Statistical Workload Analysis and Replay. Technical Report UCB/EECS-2010-81, EECS Department, University of California, Berkeley, May 2010.

[19] G. Clark, T. Courtney, D. Daly, D. Deavours, S. Derisavi, J. M. Doyle, W. H. Sanders, and P. Webster. The mobius modeling tool. In *Proceedings of the 9th international Workshop on Petri Nets and Performance Models (PNPM'01)*, pages 241–, Washington, DC, USA, 2001. IEEE Computer Society.

[20] T. Courtney, S. Gaonkar, K. Keefe, E. Rozier, and W. H. Sanders. Möbius 2.3: An extensible tool for dependability, security, and performance evaluation of large and complex system models. In *DSN*, pages 353–358. IEEE, 2009.

[21] J. Dai, J. Huang, S. Huang, B. Huang, and Y. Liu. HiTune: dataflow-based performance analysis for Big Data cloud. In *Proceedings of the 3rd USENIX conference on Hot topics in cloud computing*, HotCloud'11, pages 24–24, Berkeley, CA, USA, 2011. USENIX Association.

[22] G. Franceschinis, M. Gribaudo, M. Iacono, S. Marrone, F. Moscato, and V. Vittorini. Interfaces and binding in component based development of formal models. In *Proceedings of the Fourth International ICST Conference on Performance Evaluation Methodologies and Tools*, VALUETOOLS '09, pages 44:1–44:10, ICST, Brussels, Belgium, Belgium, 2009. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[23] Y. Fu, H. Jiang, and N. Xiao. A scalable inline cluster deduplication framework for Big Data protection. In *Proceedings of the 13th International Middleware Conference*, Middleware '12, pages 354–373, New York, NY, USA, 2012. Springer-Verlag New York, Inc.

[24] M. Iacono, E. Barbierato, and M. Gribaudo. The SIMTHESys multiformalism modeling framework. *Computers and Mathematics with Applications*, (64):3828–3839, 2012.

[25] M. Iacono and M. Gribaudo. Element based semantics in multi formalism performance models. In *MASCOTS*, pages 413–416. IEEE, 2010.

[26] IBM, P. Zikopoulos, and C. Eaton. *Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data*. McGraw-Hill Osborne Media, 1st edition, 2011.

[27] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. *SIGOPS Oper. Syst. Rev.*, 41(3):59–72, mar 2007.

[28] D. Kartson, G. Balbo, S. Donatelli, G. Franceschinis, and G. Conte. *Modelling with Generalized Stochastic Petri Nets*. John Wiley & Sons, Inc., New York, NY, USA, 1994.

[29] S. Madden. From databases to Big Data. *IEEE Internet Computing*, 16(3):4–6, 2012.

[30] MongoDB. MongoDB web site, 2011.

[31] Oozie. Oozie web site, 2011.

[32] Y. Shi, X. Meng, J. Zhao, X. Hu, B. Liu, and H. Wang. Benchmarking cloud-based data management systems. In *Proceedings of the second international workshop on Cloud data management*, CloudDB '10, pages 47–54, New York, NY, USA, 2010. ACM.

[33] V. Vittorini, M. Iacono, N. Mazzocca, and G. Franceschinis. The OsMoSys approach to multi-formalism modeling of systems. *Software and System Modeling*, 3(1):68–81, 2004.

[34] T. White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 1st edition, 2009.

[35] Y. Wu, G. Li, L. Wang, Y. Ma, J. KoÅĆodziej, and S. U. Khan. A review of data intensive computing. In *The 12th IEEE International Conference on Scalable Computing and Communications (ScalCom 2012)*. IEEE, dec 2012.

[36] Z. Zhang, L. Cherkasova, and B. T. Loo. Benchmarking approach for designing a mapreduce performance model. In *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*, ICPE '13, pages 253–258, New York, NY, USA, 2013. ACM.