Future Generation Computer Systems 37 (2014) 345–353

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Performance evaluation of NoSQL big-data applications using multi-formalism models

Enrico Barbierato [a], Marco Gribaudo [b,*], Mauro Iacono [c]

[a] DI, Università degli Studi di Torino, corso Svizzera, 185, 10129 Torino, Italy
[b] DEI, Politecnico di Milano, via Ponzio 34/5, 20133 Milano, Italy
[c] DSP, Seconda Università degli Studi di Napoli, viale Ellittico 31, 81100 Caserta, Italy

HIGHLIGHTS

- We model performances of Apache Hadoop based Big Data applications using HQL.
- Evaluation exploits the multiformalism modeling and custom domain description languages.
- The approach simplifies modeling, as it uses HQL queries as model components.
- Solution is computed by means of simulation of the aggregated model components.

ARTICLE INFO

Article history:
Received 15 July 2013
Received in revised form
9 December 2013
Accepted 27 December 2013
Available online 20 January 2014

Keywords:
Multiformalism modeling
Big data
Performance evaluation

ABSTRACT

Starting with the birth of Web 2.0, the quantity of data managed by large-scale web services has grown exponentially, posing new challenges and infrastructure requirements. This has led to new programming paradigms and architectural choices, such as map-reduce and NoSQL databases, which constitute two of the main peculiarities of the specialized massively distributed systems referred to as Big Data architectures. The underlying computer infrastructures usually face complexity requirements, resulting from the need for efficiency and speed in computing over huge evolving data sets. This is achieved by taking advantage from the features of new technologies, such as the automatic scaling and replica provisioning of Cloud environments. Although performances are a key issue for the considered applications, few performance evaluation results are currently available in this field. In this work we focus on investigating how a Big Data application designer can evaluate the performances of applications exploiting the Apache Hive query language for NoSQL databases, built over a Apache Hadoop map-reduce infrastructure.

This paper presents a dedicated modeling language and an application, showing first how it is possible to ease the modeling process and second how the semantic gap between modeling logic and the domain can be reduced, by means of vertical multiformalism modeling.

1. Introduction

Collecting huge quantities of data from the environment, from users' behavior or from massively produced contents have enabled a new perspective in information intensive applications. For instance, most of the core business of companies like Google or Facebook consists of processing data obtained from users to extract valuable information that can be sold to investors, advertisers or other third parties. The ability to create this value depends on the efficiency by which processes are performed, as computing and storage costs per data unit must be reduced (there is no guarantee that processing will produce valuable results, differently from what happens in typical data warehousing applications) and results should be available promptly (as advertisement or recommendations are only significant if provided when needed).

Performance modeling allows developers and administrators to take informed decisions, keeping efficiency high and saving time and experimental work. Designing and evaluating suitable models for these systems is not straightforward, since the number of involved computing nodes is high and can sensibly change during the lifetime of the system. Business can easily require adaptation and data dynamics are very variable. Performance modeling requires specialized expertise, given the complexity of the architectures and the interactions of Big Data oriented environments.

A dedicated language allows domain experts to ignore the methods used by evaluation tools (analytic techniques, simulations, or variants optimized for the peculiar application). It increases also the focus on the analysis process and its results, rather than on the representation of the system by intermediate description

* Corresponding author. Tel.: +39 02 2399 3568.
E-mail addresses: enrico.barbierato@mfn.unipmn.it (E. Barbierato), gribaudo@elet.polimi.it (M. Gribaudo), mauro.iacono@unina2.it (M. Iacono).

0167-739X/$ – see front matter © 2014 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.future.2013.12.036

languages (e.g. Petri nets, which would require a complete change of perspective, or simulation environment specifications, which would require a deep knowledge of the specific environment and its libraries).

On one hand, a dedicated language does not enable per se the solution of the models it defines. On the other hand, the good language design and the choice of a convenient foundation can definitely solve both the instances of modelers and solver designers. The originality of this approach consists of (i) the definition of stochastic models, able to represent the complexity of Big Data applications and architectures, (ii) the ability to encompass the problems due to the scalability of a map-reduce pipeline over a high number of nodes and the considerable amount of data involved and (iii) the possibility to minimize the semantic gap between the system and the model. Typically, suitable solution methods suffer the state space explosion effect. First, this is caused by the number of configurations of variables of the model, if based on state space oriented analytical techniques. Second, the solution may require a long and complex specification of all required elements of the system, if based on generic architectural simulation approaches.

The use of a dedicated architecture oriented language allows the automatic solution of higher level domain oriented models by means of model transformation. In this paper a vertical multiformalism approach is adopted. This includes a real world query language, which is used as high level formalism while the proposed language is used as low level formalism, in a vertical multiformalism [1].

Although the present case study is solved by a simulation engine, this paper focuses on the presentation of the proposed model description language, rather than on the proposal of a solution technique. This does not constitute a limitation, because the literature offers appropriate frameworks supporting language development and solution process management. A proposal for a solution technique based on the same approach can be found in [2].

The originality of this paper consists of an extension of [3], which presented a practical application based on a typical Big Data architecture. The reference architecture is based on Apache Hive, a data warehouse oriented DBMS relying on Apache Hadoop for executing queries written in the Hive Query Language (HQL). The extension introduces new formalism elements in the formalism from [3] and shows how it is possible to automatically generate performance models from generic HQL queries Such results can be included in multiformalism models [4], that use directly the Hive queries to describe the data-warehousing part of the system.

In particular, the advantages of multiformalism modeling are the following: it allows to describe each component of a model using the most suited description language, and it allows to automatically hide the implementation details required to make different specification interface one-another. The proposed solution has a great impact on the modeling process, giving advantages in terms of flexibility of description (either using a low-level formalism, or a high-level query), and by allowing to exploit different competency to describe separate components of the system. To the best of our knowledge there are no other comprehensive approaches to performance prediction based on formal models to data-warehousing problems such the ones considered in this paper.

The paper is organized as follows: Section 2 presents related works; Section 3 is dedicated to the description of the modeling language, and is followed by an example in Section 4; Section 5 deals with the analysis of HQL queries; the final section provides the conclusions and future work.

2. Related work

2.1. Big Data performance evaluation

Big Data is a common expression used to define the application field in which very large, generally unstructured, non relational databases must be analyzed, managed, organized and finally used to support a business. The importance of this theme, whose impetus has been given by the main industrial actors in the field of computing, is widely recognized by analysts and economists (e.g. see [5]), as well as research and academia. Big Data poses important research challenges, with reference to functional and non functional specifications on data and processes.

Big Data applications sustainability, profitability and exploitation include the following challenges: (i) scalability of computing and data storage architecture and algorithms; (ii) querying and processing technologies (including data organization and system management); (iii) planning techniques and tools and finally (iv) fault tolerance. With respect to these themes, a good introduction is given in [6,7], and a good presentation of the main themes is given in [8–11]. From the architectural point of view Apache Hadoop [12,13] appears to be the main reference, even if other approaches are available, such as domain-specific languages (DSL), which play an important role in modeling where a specific representation of a problem is requested. Dryad [14] is a general-purpose distributed execution engine working on coarse-grain data-parallel applications. It builds a dataflow graph application using a set of computational vertices and communication channels. The application executes the vertices of the graph on a set of computers, which can exchange data using shared-memory queues and TCP pipes. Oozie [15] is a server-based Workflow Engine used to run workflow jobs including control flow nodes and action nodes allowing the execution of Hadoop map-reduce jobs. Finally, NoSQL databases such as MongoDB [16] and Apache Cassandra [17] address the issue raised by relational databases regarding scalability, high availability and performance.

The literature presents some relevant contributions solving part of the architectural problems. In [18] a solution for efficient data transfer is presented, based on the RDMA over Converged Ethernet protocol, including a presentation of some of the main issues about the theme. In [19] the routing problem is presented together with a comprehensive related work section about adaptive routing and special reference to the needs of the map-reduce paradigm. The problem of protection in Big Data systems by means of cluster de-duplication is examined in [9], to support compliance to QoS parameters. Finally, [20] presents a method to exploit cloud computing infrastructures to optimize Big Data analytics applications. All of these papers provide a good insight on practical approaches to the problem and some reference measures or models, although they require a specific mathematical background beyond the common expertise of practitioners and professionals.

In [21], the authors present a performance analysis approach, based on monitoring tools and on a dataflow-driven technique. It helps designers and administrators in fine-tuning Big Data cloud environments. This approach seems very sound and comprehensive. It implicitly presents some ideas for a description of the system by means of a graphical language although it is meant for a posteriori analysis of the behavior of existing applications, rather than the supporting design. A sophisticated synthetic workload generator for map-reduce applications over cloud architectures is described in [22], which is used to evaluate performance trade-offs. It can be a significant support tool for other modeling methods. In [23] a system for benchmarking cloud-based data management systems is presented, giving useful hints on the storage performance in the most popular Big Data environments. The new trends in cloud architectures for Big Data pass the boundaries of the isolated datacenter to allow the biggest players on the market of cloud provisioning to scale up with distributed and federated cloud, posing new challenges to designers and evaluators [24–26].

2.2. Frameworks for the design of modeling languages

The main approach in the literature for the definition of custom models and modeling formalisms is based on metamodeling [27–29], a consolidated conceptual tool that has been successfully

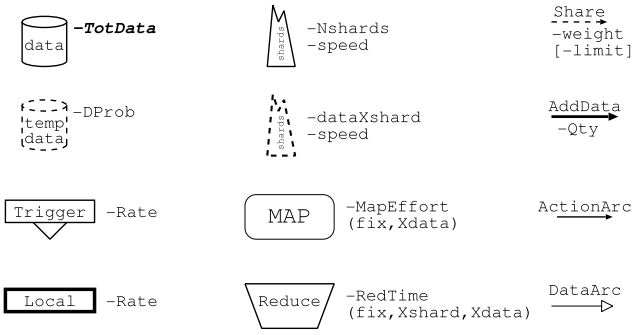

Fig. 1. Elements of the SIMTHESysBigData formalism.

exploited in several cases (e.g. Model Driven Engineering [30], software engineering [31] and multiformalism modeling [32–35]).

One of the most widespread metamodeling approaches is given by the eCore [36] framework, on which the Eclipse Modeling Framework is founded. eCore is a metamodeling stack used to enable the description of user-defined software entities, abstracting them from any detail that is related to the hardware/software platform on which they are meant to be implemented. An eCore model is an abstract definition of an application from an object-oriented approach. This includes the high level detail describing its business logic, its architecture and the relations between the objects composing the software. Such a model is used to generate automatically the equivalent source code, in a programming language chosen by the user and for the specific execution environment. To obtain this result, the eCore stack bases its models (application descriptions) on a set of modeling primitives (the eCore metamodel) designed to describe a generic object oriented software development language. Finally, it uses different model transformations (one per target environment) to generate code.

A similar metamodeling-based logic is used in the literature by two different frameworks, OsMoSys and SIMTHESys. Both support the implementation of multiformalism performance modeling techniques. The OsMoSys framework [37,32,38–43] aims to provide a tool built models and reusable model libraries. In OsMoSys, metamodeling offers the description infrastructure to describe different formal languages, with object oriented features for both models and formalisms (modeling languages), extensibility of the set of formalisms and model compositionality. OsMoSys uses a metaformalism (metametamodel) to describe any graph-based formalism, by specifying it in terms of elements (nodes) and arcs that are then specialized by each formalism. Formalisms are used to describe model classes, which describe families of models with a common structure. Elements, arcs and model classes can have properties, which form their data structure and are defined by the formalism developer and the model developer. Model classes are used to obtain reusable model class libraries, and are meant to be instantiated with actual parameters to obtain an evaluable model (model object). The OsMoSys metaformalism allows sophisticated features, including element and formalism inheritance, element hiding, definition of model interfaces, model composition and aggregation. Element, formalism and model class reuse and extension are then possible, following the object oriented general logic. The OsMoSys framework supports the definition of (multiformalism) models evaluation by means of existing external solvers, which are executed according to a business process that is related to the structure of the model and the relations between its parts.

The SIMTHESys framework [44–47,35,48–50,3] aims to provide a tool for the rapid development of new formalisms and the automatic generation of related (multiformalism) solvers. Similar to OsMoSys, it defines a metaformalism used to define formalisms, but it presents many differences. First, the SIMTHESys metaformalism defines formalism elements (indifferently nodes and arcs of a graph) having not only properties, but also behaviors describing how elements interact with each other. This allows the specification of the evaluation semantics of formalisms, besides their syntactic structure. Due to this fact, the SIMTHESys metaformalism enables each formalism to implicitly specify how models using it should be evaluated. This is exploited to generate automatically dedicated solvers for model families using a certain set of formalisms, simply applying the behavioral definitions to a selected set of solving engines (elementary non-specialized solvers implementing common base evaluation techniques). Second, in this case formalisms are directly used to define models. Third, the research effort in SIMTHESys is oriented towards automatic solver generation and formalism integration, rather than in providing sophisticated (e.g. OsMoSys object orientation of formalisms and models) modeling characteristics.

3. The SIMTHESysBigData modeling language

Without losing generality in the approach, the reference architecture on which the modeling language has been designed represents the ecosystem based on Apache Hadoop. The SIMTHESysBigData modeling language is founded on the SIMTHESys framework, which has been chosen because it offers a flexible choice of the final model solution technique, and because it is more suited to the goals of this research, as it is oriented to the specification of domain-oriented high level formalisms and allows decoupling between modeling abstractions and solution engines.

Hadoop is an implementation of the map-reduce paradigm, previously introduced by Google to manage its applications. The map-reduce paradigm is designed to cope with massively distributed execution of computing tasks with high efficiency, in order to meet the needs of Big Data applications. According to this paradigm, data is organized in a NOSQL structure over data nodes, namely *shards*. Each shard contains table-like structures, each row of which can have a fixed or a variable number of fields, differently from what happens in a relational data base organization. Shards store only a certain number of rows, dimensioned to balance the workload over the system, and each row can be addressed by a key-value based index. To perform a computation, a pipeline consisting of a sequence of stages is set up. Each stage consists of a *map phase* and a *reduce phase*. The former triggers the run of a user-defined code that is sent automatically to all involved shards. The latter efficiently collects and synthesizes the outputs to get to the final result or accounts for the completion of the operations that had to be performed on each shard.

To represent the main elements of the paradigm, the modeling language has been designed to offer the elements shown in Fig. 1.

The available elements are divided into two groups called respectively the *Structural elements* and the *Operational elements*. The former are the elements forming the structure of the architecture, specifically (i) *Dataset* representing a logical/physical group of data; (ii) *Shards* representing a group of shards, which is put in relation with a Dataset, and on which Dataset data are mapped, (iii) the *Temporary Dataset* accounting for temporary tables generated during map-reduce tasks.

The operational elements model the sequence of operations in a map-reduce pipeline and consist of (i) *Trigger* representing a data source generating data towards a Dataset with Poisson arrivals; (ii) *Map* representing a map phase in a map-reduce pipeline stage; (iii) *Reduce* representing a reduce phase in a map-reduce pipeline stage; (iv) *Local actions* representing actions that are executed locally, and that do not require the execution of a complex map-reduce task.

The available arcs are: (i) *Share* representing the binding between a logical/physical group of data and the shard on which it is allocated (and related computing is performed); (ii) *AddData* representing the binding between a Trigger and the Dataset storing the received data; (iii) *ActionArc* representing the binding between

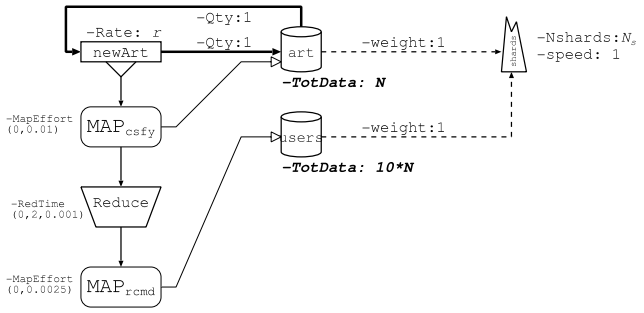

Fig. 2. Model of the example.

two temporally consequent operational elements in a map-reduce pipeline and finally (iv) *DataArc*, mapping a Map onto the Dataset on which it is applied.

Element and arc types in a model are identified by the corresponding icons shown in Fig. 1.

Within the SIMTHESys framework, each element and arc is fully specified by its properties and behaviors specifying its structure and its dynamics. In order to keep the description on a modeler the focus will be put on modeling-related properties which must be specified to evaluate an applicative scenario.

A *Dataset* element is characterized by the *TotData* property representing the current amount of contained data, since it influences the performance of the Shards element on which related computing will be performed. A *Shards* element is used to describe the shards over which data are distributed. In particular, each shard is characterized by the *Speed* property, a synthetic nominal performance indicator of a single constituting computing node. Since big-data applications are usually deployed over cloud infrastructures, two types of shards can be defined: *Fixed shards* (denoted by a continuous line) and *Auto-scaling shards* (drawn with dashed lines). The fixed shards are used to model software components distributed over a fixed number of (symmetric) computation nodes. They are characterized by the *NShards* property representing the resources over which data rows are divided. Auto-scaling shards represent computing nodes exploiting the auto-scaling features of cloud computing. In particular, they allow a dynamic deployment of virtual machines, in such a way that each shard has to deal with no more than a fixed amount of data. The property *dataXshards* specifies the maximum number of data rows that a shard can hold. *Temporary Dataset* elements represent tables that are created as intermediate steps of more complex processes, holding parts of the final computation. Since usually they are constructed by filtering an existing dataset, the *DProb* properties define the probability that a data from one of the tables handled by the corresponding event (either a map, a reduce or a local execution), generates a row in the temporary table. A *Trigger* element is characterized by the *Rate* property, representing the arrival rate of requests. To simplify the computation of performance indices, Poisson arrival processes are considered. A Map element is characterized by the *MapEffort* property describing the complexity of the map operation. In particular, this parameter is composed of two parts: a constant term (*fix*), which is required by every operation, and another term that is proportional to the number of data that must be considered (*Xdata*). In a similar way, a *Reduce* element is characterized by the *RedTime* property describing the time required to perform a reduction. This time can be composed of a fixed part (*fix*), a component proportional to the number of shards over which the data were mapped (*Xshard*) and a part that is proportional to the quantity of data that must be considered (*Xdata*). A *Local action* performs a task not depending on the dataset, which can be characterized by its speed, specified by a property *Rate*. A *Share* arc, connects a *Dataset* to a *Shards* element. It is characterized by the *Weight* property, used to define which fraction of the total amount of data should be put over the considered shards. The optional *Limit* property, represents the maximum number of rows that can be put on each machine of the destination shards. The *AddData* arcs connect *Trigger* to *Dataset* elements. They represent an increase or decrease of data in the destination data set, and they are characterized by the *Qty* property, accounting for the (possibly negative) number of rows that are added (or removed); all other language elements have no significant modeling-related property.

The presented elements and arcs have been described in a SIMTHESys FDL (Formalism Description Language) document, to enable the design of SIMTHESys MDL (Model Description Language) documents describing models.[1]

Before the full integration of the language into the SIMTHESys framework currently in progress, the language has been used to develop an experimental simulator implementing its primitives, supporting and guiding the study of the best solution engine suitable for the field. Although the simulator is fully functional for the goals of this paper, the implemented solution is considered, in the SIMTHESys perspective, a proof of concept because the solver currently does not support the integration of multiple formalisms and the generation of the solver is not completely automated.

4. Example

To exemplify the use of the SIMTHESysBigData formalism, a scenario based on a real system is considered. Performance analysis is needed to evaluate the opportunity of a migration or architectural reconfiguration. The system is used to run an on-line content publishing system, capable of social network functionalities. The application operates by allowing a certain number of journalists to publish articles about different topics. Registered users can publish comments, or other articles as well. Articles are proposed to registered users, according to their user habits and interests, which are profiled by the application. Proposals are generated with a recommendation system. The relevant data set to be stored on the shards consists of: published articles, users' data and users' profiling data.

The recommendation functionality is implemented by a map-reduce pipeline: a first map-reduce stage classifies each new article by a comparison with all existing articles in the system, and returns a synthetic classification, that is compared by a second map-reduce stage to analogous synthetic classifications of the interests of each user, to assign recommendations.

Performance evaluation has been applied to the recommendation functionality. The model is depicted in Fig. 2.

The left part of the figure represents the map-reduce pipeline implementing the recommendation system, while the right part represents the architecture and the mapping of datasets and shards. The NewArt *Trigger* element represents the arrival of new articles, that are produced at a rate of r art/min, where r is a parameter of the study. An *ActionArc* arc connects it to the MAPcsfy *Map* element, determining when the classification map phase will be performed. Two *AddData* arc connects it to the Art *DataSet* element in the opposite direction, and both with the quantity attribute equal to one ($Qty = 1$). This is used to account for the fact that the system tries to maintain constant the number of articles, by replacing an old one (arc going out from the *DataSet* element), with the new arrival (arc going into the *DataSet* element). The number of articles N stored in the dataset is a parameter of the study. The MAPcsfy *Map* element represents the influence in the process of the map phase of the pipeline. It is connected with a *DataArc* arc to

[1] The FDL document for the language and the MDL document for the case study are omitted for the sake of space, but current versions of both can be freely obtained by sending the authors a request by email.

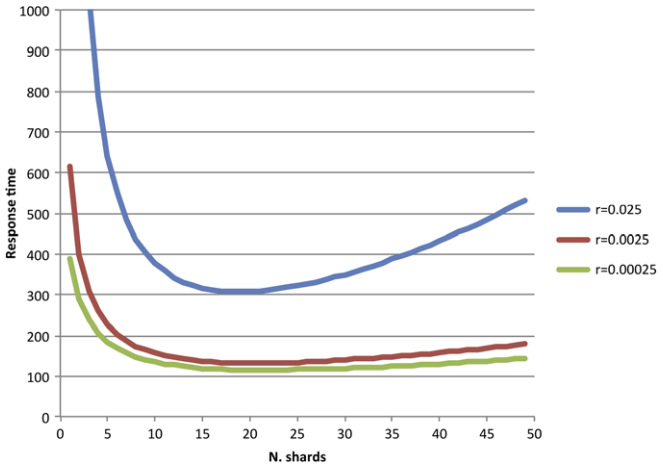

Fig. 3. Response time for a varying number of shards for different article arrival rates r.

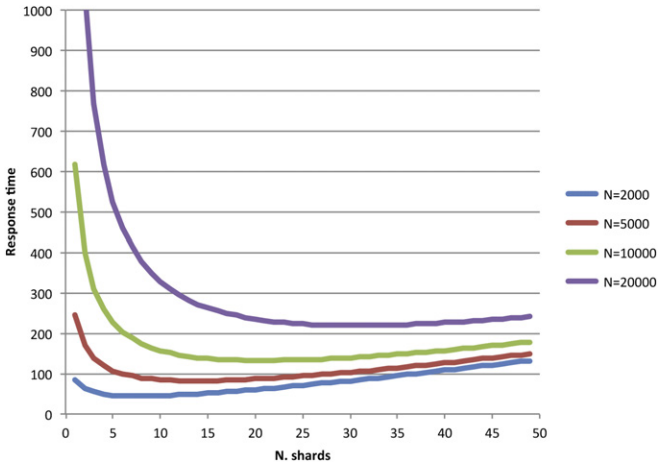

Fig. 4. Response time for a varying number of shards for different number of articles N.

the Art *DataSet* element, on which it operates, through which the performance of the shards are considered in the phase. The time required to perform this mapping is proportional to the size of the dataset, and does not have a fixed part: each element of the dataset increases the time required to perform the mapping of 0.01 min. The end of the mapping phase triggers the operations of the *Reduce* element, which in turn accounts for the contribution of the reduce phase. The time to perform the reduction requires 2 min per shard, and 0.001 min per row in the article dataset, The *Reduce* element is connected by an *ActionArc* to the *Map* element MAPrcmd performing recommendations (the related *Reduce* phase is negligible in the application and is thus omitted), in turn connected by a *DataArc* arc to the Users *Dataset* element, on which it operates. This operation requires 0.0025 min per user in the Users dataset. Both the Art and Users *Dataset* elements are connected by *Share* arcs to the Shards *Shard* element. The size of the two datasets is a parameter N of the model. In particular, it is supposed that the number of users in the system is ten time greater than the number of articles: this can be clearly seen in Fig. 2 by the value assigned to the property *TotData* of the two *DataSet* elements. The number of shards N_s is also a parameter of the model. In this case it is supposed that all the shards working at the same speed, assigned identically to 1 operation per minute: in this way the effort used to describe the map phases of the model is equal to the time required to perform that operation. Finally, all the data are equally split among the shards: this is represented by the property *weight* $= 1$ assigned to the two *Share* arcs.

In this study, the number of article is set as $N = 10\,000$, and varying the number of shards N_s, for different new article arrival rates r. The results are presented in Fig. 3. As it can be seen, the response time has a curved shape with a minimum. For a low number of shards, there is a large response time due to the high work that the few shards have to perform to complete the map operations. When the number of shards is high, the big latency for waiting all the nodes to finish their task, and the increased complexity of the reduce operations, make the response time grow again. Thus determining the best number of shards of an application can be a good motivation for using performance evaluation formalisms like the one proposed in this paper. In this case, it is observed that the number of shards giving the minimum response time ($N_s = 22$) is independent from the article arrival rate.

The effect of changing the number of articles N is studied (and thus proportionally the number of users), with a fixed arrival rate of $r = 0.0025$ art/s. Resulting response times are plotted in Fig. 4. As it can be seen, also in this case the response time has a minimum for a given number of shards. However, in this case the position of the minimum varies with the size of the dataset. In particular, larger datasets require a higher optimal number of shards. Since simulation was used to compute the presented results, 95% confidence intervals were used. However, only the mean response times were plotted in Figs. 3 and 4 to simplify the presentation. The obtained intervals were very tight, as can be seen in Table 1 for some number of shards N_s, considering an arrival rate of new articles $r = 0.0025$ art/min, and a dataset size of $N = 10\,000$.

Table 1
Confidence interval for the response time with different number of shards, for an arrival rate of new articles $r = 0.0025$ art/min, and a dataset size of $N = 10\,000$.

N_s	R^-	R	R^+
5	642.040864	642.822	643.603136
10	377.098435	377.441	377.783565
15	315.831929	316.154	316.476071
20	307.109146	307.475	307.840854

5. Modeling Apache Hive queries

Higher level tools for Big Data applications development can be found in the literature, besides low-level approaches like map-reduce, mainly relying on NoSQL databases. This section focuses on Apache Hive [51]. While Apache Hive is not properly a NoSQL database, it provides an SQL-like dialect, called Hive Query Language (HQL) for querying data stored in Hadoop clusters. This simplifies the migration from existing SQL applications, since it provides a very similar programming paradigm, providing significative scaling-up opportunities. The main purpose of Hive is to support data warehouse applications, were relatively static data are analyzed, and real-time access is not required. Hive does not deal with standard SQL tables, and it does not provide record-level update, insert and delete. Since it relies on Hadoop, which is a batch-oriented system, HQL queries have a high latency since map-reduce jobs are characterized by a large start-up overhead. However, when dealing with big data sets, this overhead becomes negligible compared to the total processing time.

Data is organized along *partitioned tables*: each table is split across several files. The values of some fields (called *partition keys*) can be used to create a file name, which identifies the portion of the table. Each file contains all the records characterized by the same values assigned to the considered partition keys.

5.1. Modeling HQL queries

Since HQL queries are executed by map-reduce tasks, they can be easily modeled with the custom formalism presented in

Section 3. Specifically, HQL includes the *EXPLAIN* command listing the Map and Reduce jobs required to execute a query. A model for the query can then be directly produced, by converting the tasks identified by the query plan, with the equivalent model primitives.

Let us focus on one of the Hive queries commonly used as a benchmarking application. Let us imagine that the considered DB includes a table, named *docs*, containing a set of lines extracted form a corpus of texts. The records of the table are composed by a single string field called *line*, containing one line of the considered documents. The query used to perform the word count is the following [52]:

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

The query performs the word count in the following way: as first, it considers all the lines stored in the *docs table*, and extracts the single words with the *split()* function. Each element returned from the splitting procedure is transformed into a different row using the *explode()* function. The next query counts identical words using the *count()* function and the *GROUP BY* statement. Results are then ordered by word, thanks to the *ORDER BY* clause, and finally inserted into a new table called word_counts by using the *CREATE TABLE* statement. By prefixing the considered query with the *EXPLAIN* command, a query plan follows, which is composed of local actions, map and reduce operations.

The query is performed in five stages, named *Stage*-0 to *Stage*-4. In Hive, stages are numbered according to the position in which they appear in the query, rather than in the order of execution. The order can be derived from the dependency information, which specifies which stages need to be completed before the considered one can start. This allows to include parallelism into queries, when some stages have no internal dependencies. For the considered example, there is a fixed order (where no parallelism is allowed), and stages are executed in the following order:

Stage-1 → *Stage*-2 → *Stage*-0 → *Stage*-4 → *Stage*-3.

Stages that can be performed independently at the beginning of the process are called *root stages*: the considered query has only one root stage (*Stage*-1).

The 5 stages are the following. The first stage of the query does the biggest part of the job, and it is executed via a map-reduce operation. The map phase considers the lines of the text stored in the DB, and extract the words contained there, by executing both the *split()* and *explode()* commands. The grouping function performing the actual word count is executed during the reduce stage. The output is an un-sorted list of words, together with an integer field containing the number of occurrences. The output is stored in a temporary table, which is then used by the following stages of the query. The second stage sorts the counts according to the corresponding words: it is also performed by a map-reduce task working on the temporary table created during the first stage. The map part just executes an identity function, while the actual sorting is performed during the reduce. Results are stored in another temporary table. The third stage renames the temporary table generated in the previous stage to its final name (*word_count*). The last two stages perform integrity operations: the fourth one updates the schema repository to include the new table, and the last one updates the statistics over the table involved in the query to allow better optimizations of generated query plans.

HQL queries can be easily translated in the proposed Big Data formalism to study their performance. In particular, map-reduce stages are converted into a sequence of a map node and a reduce node, while other type of stages are converted to local actions,

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

Fig. 5. The HQL to Big Data model translation algorithm.

```
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
```

Fig. 6. The HQL to Big Data model translation algorithm: elements generation.

since they do not start tasks on the distributed infrastructure. Stages can be characterized by both input and output tables, which can be either temporary or permanent. Tables are modeled by *Dataset* or *Temporary Dataset* primitives, depending on whether they are permanent or not. Input table of a map-reduce operation are connected via *Data arc*s, since they provide the data on which the stage operates. Data generated by the stage are inserted in the corresponding table via *AddData* arcs. If a temporary table is later moved to a permanent table (such as the table containing the word counts in the considered example), it is modeled directly with a *Dataset* instead of a *Temporary Dataset*. All stages not involving map-reduce are converted to *Local actions*. A *Trigger* element is added to model the event starting the execution of the query. The sequence of stages determines the connection among the Big Data formalism primitives. From each *Map* element departs an *ActionArc* connecting it to the corresponding *Reduce* element. A set of *ActionArcs* connects the *Trigger* element representing the execution of the query with the nodes corresponding to each *root* stage. Then another *ActionArc* is added for each non-root stage: this arc connects them to the stages from which it depends. *Datasets* (both normal and temporary) are finally connected with the *Shards* modeling the physical infrastructure of the system with *Share* arcs.

The complete translation algorithm is presented in Fig. 5. Command `NEW BigDataModel()` creates a new empty Big Data model, and command `ExecuteQuery()`, executes the *EXPLAIN* command on the query that is being translated and stores its result in variable `$expQ`. The algorithm first convert the stages by calling the `TranslateQuery()` procedure shown in Fig. 6, and then connects the stages with the `ConnectModel()` procedure presented in Fig. 7.

Stage conversion, first considers the map-reduce stages and adds the corresponding elements to the model (Fig. 6). Method `$model.add($etype, $id)`, adds an element of type `$etype` to

```
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
```

Fig. 7. The HQL to Big Data model translation algorithm: connect stages.

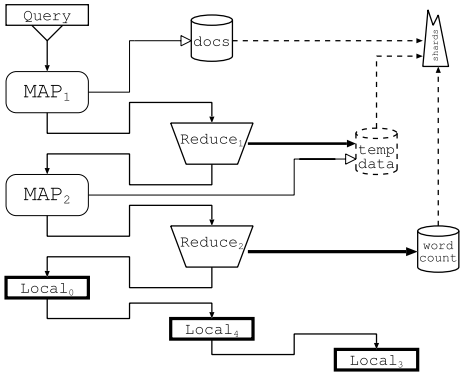

Fig. 8. Model generated by the considered word-count query.

the model with the given $id, and $model.addArc($atype, $from, $to) adds an arc of type. Methods $s.getInTables() and $s.getOutTables() return respectively the tables used in the considered stage $s as either input or output, while method $t.isTemporary() checks if a table is temporary. The $t.isInput() checks whether a table is input or output of the considered map-reduce stage, and it is used to decide which type of arc must be created. After considering the map-reduce stage, the other stages are converted to local actions. In addition, if a stage is a stage moving a table (method $s.getType()), when the source is temporary, but the destination is permanent, then the source table is eliminated from the model, and replaced by the permanent table with the method model.MergeToPermanentTable().

Finally, the procedure connecting the model (Fig. 7) examines the dependencies among the stages and connects them. First it adds the *Query* element representing the triggering of the query and connects all the root stages to it. Then it considers all the non-root stages, and connects them to all the stages on which they depend.

The model of the word count query is shown in Fig. 8. As it can be seen, all the stages have been translated either to a sub-model composed by a sequence of *Map* and *Reduce* or to *Local* elements. Three tables have been generated, corresponding to the input, the output and the temporary tables used by the query. The output table (*word_count*), has been converted into a permanent table due to the presence of the *Move* stage (*Stage*-0, third in order of execution). The *Shards* element and the *Share* arcs connecting the datasets to it have been added after the query translation: this allows the separation of the logic (the HQL query), from the physical infrastructure definition where the query is run.

5.2. Query performance evaluation

To validate the proposed methodology, the word-count query on a dataset composed of a set of articles downloaded from 20 newsgroups is tested. The sole purpose of this example is to show how the proposed modeling technique can be applied to a realistic

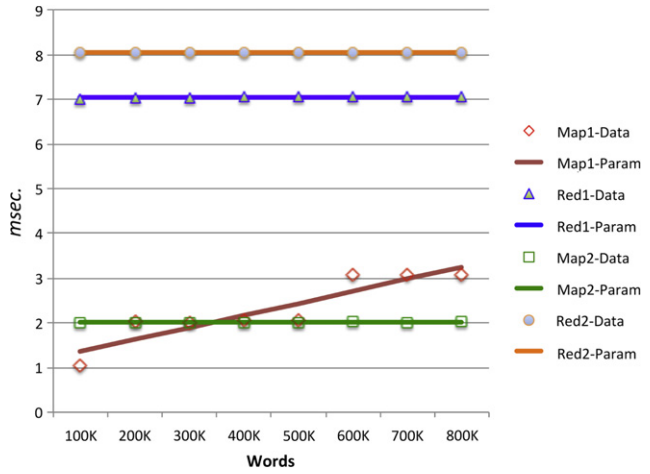

Fig. 9. Measured lengths of map-reduce stages, and the corresponding parameters used in the model.

case, and not to provide a highly efficient indexation application architecture. In particular, since the whole dataset was about 800 K words, it has been partitioned it into 8 groups of around 100 K words each. The query was repeated eight times considering every time an increased number of partitions. Experiments were run on a Tiny Core Linux virtual machine, running under VirtualBox on an Intel i5 MacBook Air. The Hadoop infrastructure was installed in Pseudo-Distributed mode, to simulate a larger infrastructure on a single-node environment. Fig. 9 shows the data measured and the parameters used in the model. The reduce of the first stage, and the second map-reduce stage have durations not depending on the size of the dataset: their durations are respectively 7.047 ms, 2.017 ms and 8.053 ms. This is due to the fact that even the smallest corpus (100 K) already includes almost the full range of commonly used words. For this reason, both the temporary table and the word-count table have more or less the same number of records in all experiments. The map of the first stage depends instead on the size of the dataset in a stepwise manner: this is due to the way in which the Hadoop file system stores the data. Hive tables are memorized in Hadoop files that are stored in very large chunks. The steps in the measured response time correspond to the time required to process the different (integer) number of chunks over which the dataset is stored. To simplify the parametrization, a fixed effort of 1.093 ms was set, plus an extra effort of 0.0027 ms per K words. The other stages have a mean length of 10.108 ms, which was arbitrarily divided as 10% for the move operation, 50% for the schema update, and 40% for the table statistics computation.

The model was used to test the scalability and define the optimal deployment strategy for an application relying on the considered query. Specifically, the effect of distributing the Hadoop infrastructure over an increasing number of nodes was tested, depending on the size of the dataset. Fig. 10 shows the results. As it can be seen, when the data set is relatively small (less than 2 M words) there is no advantage in using more than one shard. Even worse, the response time tends to increase when the infrastructure is distributed over more nodes due to the added synchronization issues. However, when the dataset becomes really huge, the system starts to experience an improvement in performance due to parallelization. For example with 5 M words, the minimum response time is obtained with 10 shards reducing the time complete the query from 42.916 to 38.312 s. With 10 M words, the optimum deployment is with 24 shards, reducing even further the response time from 55.356 to 41.229 s. These results confirm that the Big Data infrastructure can provide effective results only for extremely large datasets, while they experience poor performance when the data size is small. Modeling techniques such as the one proposed in this paper, provide significative contribution in order to identify the optimum operational point.

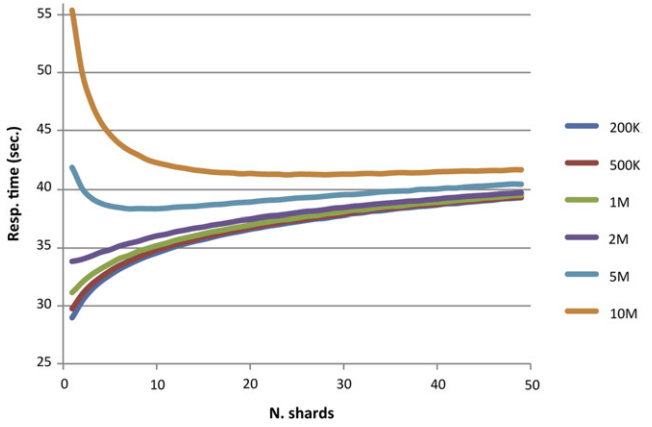

Fig. 10. Response time for different number of shards and different size of the considered dataset.

6. Conclusions and future work

This paper has presented a novel language for the description of performance models including applications based on the map-reduce paradigm. The main contribution of this work is to allow Big Data application designers and Big Data system administrators to evaluate their choices and experiment with what-if analysis, at two different abstraction levels, by means of a vertical, transformation based multiformalism approach. From the authors' evaluation, the proposed approach is suited to the task, as it represents a complex environment such as Big Data applications with a comfortable metaphor. At the same time, it is suitable for the automated generation of solvers without deep expertise, thanks to the fact that it was founded on the SIMTHESys framework.

The work is in progress to allow the language to be used to automatically generate both analytical and simulation solvers that also enable the designer to incorporate submodels specified in other modeling languages (such as Petri nets variants or Fault Trees). It will also be necessary to extend the simulator to a full version that can be fully integrated in SIMTHESys as a solving engine and can automatically handle a variable number of trigger-generated items and shards, without the user intervention on the model.

References

[1] M. Gribaudo, M. Iacono (Eds.), Theory and Application of Multi-Formalism Modeling, IGI Global, 2014.
[2] A. Castiglione, M. Gribaudo, M. Iacono, F. Palmieri, Exploiting mean field analysis to model performances of big data architectures, Future Gener. Comput. Syst. (2013). http://dx.doi.org/10.1016/j.future.2013.07.016, in press.
[3] E. Barbierato, M. Gribaudo, 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, Aalesund, Norway, 27–30 May, 2013.
[4] E. Barbierato, M. Gribaudo, M. Iacono, Modeling Apache Hive based applications in big data architectures, in: Proceedings of 7th International Conference on Performance Evaluation Methodologies and Tools—VALUETOOLS 2013, 2013.
[5] T. Economist, Drowning in numbers–digital data will flood the planet—and help us understand it better, Economist (2011). URL: http://www.economist.com/blogs/dailychart/2011/11/big-data-0.
[6] Y. Wu, G. Li, L. Wang, Y. Ma, J. Kolodziej, S.U. Khan, A review of data intensive computing, in: The 12th IEEE International Conference on Scalable Computing and Communications, ScalCom 2012, IEEE, 2012.
[7] S. Madden, From databases to big data, IEEE Internet Comput. 16 (3) (2012) 4–6.
[8] E. Bertino, P. Bernstein, D. Agrawal, S. Davidson, U. Dayal, M. Franklin, J. Gehrke, L. Haas, A. Halevy, J. Han, et al. Challenges and opportunities with big data, 2011.
[9] Y. Fu, H. Jiang, N. Xiao, A scalable inline cluster deduplication framework for big data protection, in: Proceedings of the 13th International Middleware Conference, Middleware'12, Springer-Verlag New York, Inc., New York, NY, USA, 2012, pp. 354–373. URL: http://dl.acm.org/citation.cfm?id=2442626.2442649.
[10] R.E. Bryant, R.H. Katz, 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.
[11] D. deRoos, C. Eaton, G. Lapis, P. Zikopoulos, T. Deutsch, Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data, first ed., McGraw-Hill Osborne Media, 2011.
[12] Apache Hadoop, Apache Hadoop web site, 2008. URL: http://hadoop.apache.org/.
[13] T. White, Hadoop: The Definitive Guide, first ed., O'Reilly Media, Inc., 2009.
[14] M. Isard, M. Budiu, Y. Yu, A. Birrell, D. Fetterly, Dryad: distributed data-parallel programs from sequential building blocks, in: Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007, EuroSys'07, ACM, New York, NY, USA, 2007, pp. 59–72. URL: http://doi.acm.org/10.1145/1272996.1273005.
[15] Oozie, Oozie web site, 2011. URL: http://oozie.apache.org/.
[16] MongoDB, MongoDB web site, 2011. URL: http://www.10gen.com/products/mongodb.
[17] Apache Cassandra, Apache Cassandra web site, 2009. URL: http://cassandra.apache.org/.
[18] B. Tierney, E. Kissel, D.M. Swany, E. Pouyoul, Efficient data transfer protocols for big data, in: eScience, IEEE Computer Society, 2012, pp. 1–9.
[19] E. Zahavi, I. Keslassy, A. Kolodny, Distributed adaptive routing for big-data applications running on data center networks, in: Proceedings of the Eighth ACM/IEEE Symposium on Architectures for Networking and Communications Systems, ANCS'12, ACM, New York, NY, USA, 2012, pp. 99–110. URL: http://doi.acm.org/10.1145/2396556.2396578.
[20] G. Jung, N. Gnanasambandam, T. Mukherjee, Synchronous parallel processing of big-data analytics services to optimize performance in federated clouds, in: Proceedings of the 2012 IEEE Fifth International Conference on Cloud Computing, CLOUD'12, IEEE Computer Society, Washington, DC, USA, 2012, pp. 811–818. URL: http://dx.doi.org/10.1109/CLOUD.2012.108.
[21] J. Dai, J. Huang, S. Huang, B. Huang, 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, USENIX Association, Berkeley, CA, USA, 2011, p. 24. URL: http://dl.acm.org/citation.cfm?id=2170444.2170468.
[22] Y. Chen, A.S. Ganapathi, R. Griffith, R.H. Katz, Towards Understanding Cloud Performance Tradeoffs Using Statistical Workload Analysis and Replay, Tech. Rep. UCB/EECS-2010-81, EECS Department, University of California, Berkeley, 2010. URL: http://www.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-81.html.
[23] Y. Shi, X. Meng, J. Zhao, X. Hu, B. Liu, H. Wang, Benchmarking cloud-based data management systems, in: Proceedings of the Second International Workshop on Cloud Data Management, CloudDB'10, ACM, New York, NY, USA, 2010, pp. 47–54. URL: http://doi.acm.org/10.1145/1871929.1871938.
[24] L. Wang, J. Tao, R. Ranjan, H. Marten, A. Streit, J. Chen, D. Chen, G-Hadoop: MapReduce across distributed data centers for data-intensive computing, Future Gener. Comput. Syst. 29 (3) (2013) 739–750. Special Section: Recent Developments in High Performance Computing and Security. URL: http://www.sciencedirect.com/science/article/pii/S0167739X12001744.
[25] D. Chen, L. Wang, X. Wu, J. Chen, S.U. Khan, J. Kolodziej, M. Tian, F. Huang, W. Liu, Hybrid modelling and simulation of huge crowd over a hierarchical grid architecture, Future Gener. Comput. Syst. 29 (5) (2013) 1309–1317. Special section: Hybrid Cloud Computing. URL: http://www.sciencedirect.com/science/article/pii/S0167739X12000593.
[26] W. Lizhe, J. Tao, Y. Ma, S.U. Khan, J. Kolodziej, D. Chen, Software design and implementation for map-reduce across distributed data centers, Appl. Math. Inf. Sci. 7 (1) (2013) 85–90.
[27] J. Bézivin, On the unification power of models, Softw. Syst. Model. 4 (2) (2005) 171–188.
[28] J.P. Van Gigch, in: John P. van Gigch (Ed.), System Design Modeling and Metamodeling, Plenum Press, New York, 1991.
[29] M.A. Jeusfeld, M. Jarke, J. Mylopoulos (Eds.), Metamodeling for Method Engineering, MIT Press, Cambridge, MA, USA, 2009.
[30] J.D. Poole, Model-driven architecture: vision, standards, and emerging technologies, Position Paper ECOOP 2001, 2001, submitted for publication.
[31] Object Management Group, Unified modeling language standards version 2.3, 2010. URL: http://www.omg.org/spec/UML/2.3/.
[32] V. Vittorini, M. Iacono, N. Mazzocca, G. Franceschinis, The OsMoSys approach to multi-formalism modeling of systems, Softw. Syst. Model. 3 (1) (2004) 68–81.
[33] H.M. Gholizadeh, M.A. Azgomi, A meta-model based approach for definition of a multi-formalism modeling framework, Int. J. Comput. Theory Eng. 2 (1) (2010) 87–95.
[34] J. de Lara, H. Vangheluwe, Atom3: a tool for multi-formalism and meta-modelling, in: R.-D. Kutsche, H. Weber (Eds.), FASE, in: Lecture Notes in Computer Science, vol. 2306, Springer, 2002, pp. 174–188.
[35] M. Iacono, E. Barbierato, M. Gribaudo, The SIMTHESys multiformalism modeling framework, Comput. Math. Appl. (64) (2012) 3828–3839.
[36] D. Steinberg, F. Budinsky, M. Paternostro, E. Merks, EMF: Eclipse Modeling Framework, second ed., Addison-Wesley Professional, 2008.
[37] G. Gribaudo, M. Iacono, M. Mazzocca, V. Vittorini, The OsMoSys/DrawNET Xe! Languages system: a novel infrastructure for multi-formalism object-oriented modelling, in: ESS 2003: 15th European Simulation Symposium and Exhibition, 2003.

[38] F. Moscato, F. Flammini, G.D. Lorenzo, V. Vittorini, S. Marrone, M. Iacono, The software architecture of the OsMoSys multisolution framework, in: ValueTools'07: Proceedings of the 2nd International Conference on Performance Evaluation Methodologies and Tools, 2007, pp. 1–10.
[39] G. Franceschinis, M. Gribaudo, M. Iacono, S. Marrone, N. Mazzocca, V. Vittorini, Compositional modeling of complex systems: contact center scenarios in OsMoSys, in: ICATPN'04, 2004, pp. 177–196.
[40] V. Vittorini, G. Franceschinis, M. Gribaudo, M. Iacono, N. Mazzocca, DrawNet++: model objects to support performance analysis and simulation of complex systems, in: Proc. of the 12th Int. Conference on Modelling Tools and Techniques for Computer and Communication System Performance Evaluation, TOOLS 2002, London, UK, 2002.
[41] F. Franceschinis, M. Gribaudo, M. Iacono, N. Mazzocca, V. Vittorini, Towards an object based multi-formalism multi-solution modeling approach, in: Daniel Moldt (Ed.), Proc. of the Second International Workshop on Modelling of Objects, Components, and Agents, MOCA'02, Aarhus, Denmark, August 26–27, 2002, Technical Report DAIMI PB-561, 2002, pp. 47–66.
[42] G. Franceschinis, M. Gribaudo, M. Iacono, V. Vittorini, C. Bertoncello, DrawNet++: a flexible framework for building dependability models, in: Proc. of the Int. Conf. on Dependable Systems and Networks. Washington DC, USA, 2002.
[43] G. Franceschinis, M. Gribaudo, M. Iacono, S. Marrone, F. Moscato, 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, ICST, Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, Brussels, Belgium, Belgium, 2009, pp. 44:1–44:10.
[44] E. Barbierato, M. Gribaudo, M. Iacono, Defining formalisms for performance evaluation with SIMTHESys, Electron. Notes Theor. Comput. Sci. 275 (2011) 37–51.
[45] E. Barbierato, M. Gribaudo, 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.
[46] E. Barbierato, M. Gribaudo, M. Iacono, S. Marrone, Performability modeling of exceptions-aware systems in multiformalism tools, in: K. Al-Begain, S. Balsamo, D. Fiems, A. Marin (Eds.), ASMTA, in: Lecture Notes in Computer Science, vol. 6751, Springer, 2011, pp. 257–272.
[47] M. Iacono, M. Gribaudo, Element based semantics in multi formalism performance models, in: MASCOTS, IEEE, 2010, pp. 413–416.
[48] E. Barbierato, A. Bobbio, M. Gribaudo, M. Iacono, Multiformalism to support software rejuvenation modeling, in: ISSRE Workshops, IEEE, 2012, pp. 271–276.
[49] E. Barbierato, M. Iacono, S. Marrone, PerfBPEL: a graph-based approach for the performance analysis of BPEL SOA applications, in: VALUETOOLS, IEEE, 2012, pp. 64–73.
[50] E. Barbierato, G.D. Rossi, M. Gribaudo, M. Iacono, A. Marin, Exploiting product form solution techniques in multiformalism modeling, Electron. Notes Theor. Comput. Sci. (2012) in press.
[51] Apache Hive, Apache Hive web site, 2013. URL: http://hive.apache.org/.
[52] E. Capriolo, D. Wampler, J. Rutherglen (Eds.), Programming Hive—Data Warehouse and Query Language for Hadoop, O'Reilly Media, 2012.

Enrico Barbierato is a Consultant working for the IT industry. He earned a B.Sc. (Hon) from the University of Turin (Italy), an M.Sc. in Advanced Studies in Artificial Intelligence from the Katholieke Universiteit of Leuven (Belgium) and a Ph.D. in Computer Science from the University of Turin (Italy). His research activity concerns the performance evaluation of multiformalism models.

Marco Gribaudo is a senior researcher at the Politecnico di Milano—Italy. He works in the performance evaluation group. His current research interests are multi-formalism modeling, queueing networks, mean-field analysis and spa-tial models. The main applications to which the previous methodologies are applied come from cloud computing, multi-core architectures and wireless sensor networks.

Mauro Iacono is a tenured Assistant Professor and Senior Researcher in Computing Systems at the "Dipartimento di Scienze Politiche" department of "Seconda Università degli Studi di Napoli", Caserta, Italy. He received the Laurea in Ingegneria Informatica (M.Sc.) degree with full grades and honors (cum laude) in 1999 by Universitadegli Studi di Napoli "Federico II", Napoli, Italy, and the Dottorato in Ingegneria Elettronica (Ph.D.) degree by "Seconda Universitá degli Studi di Napoli", Aversa, Italy. He published over 30 peer reviewed scientific papers in international journals and conferences and has served as scientific editor, conference scientific committee chairman and member and reviewer for several journals, and is a member of IEEE and other scientific societies. His research activity is mainly centered on the field of performance modeling of complex computer-based systems, with a special attention for multiformalism modeling techniques. More information is available at www.mauroiacono.com.