A PERFORMANCE MODELING LANGUAGE FOR BIG DATA ARCHITECTURES

Enrico Barbierato
Dipartimento di Informatica
Università degli Studi di Torino
corso Svizzera 185
10129 Torino, Italy
Email: enrico.barbierato@mfn.unipmn.it

Marco Gribaudo
Dipartimento Di Elettronica,
Informazione E Bioingegneria
Politecnico di Milano
via Ponzio 51
20133, Milano, Italy
Email: gribaudo@elet.polimi.it

Mauro Iacono
Dipartimento di Scienze Politiche
Seconda Università degli Studi di Napoli
viale Ellittico 31
81100, Caserta, Italy
Email: mauro.iacono@unina2.it

KEYWORDS

Big Data; performance modeling; modeling tools; metamodeling

ABSTRACT

Big Data applications represent an emerging field, which have proved to be crucial in business intelligence and in massive data management. Big Data promises to be the next big thing in the development of strategical computer applications, even if it requires considerable investment and an accurate resource planning, as the architectures needed to perform at the requisite speed need to scale easily on to a large number of computing nodes. Appropriate management of such architectures benefits from the availability of performance models, to allow developers and administrators to take informed decisions, saving time and experimental work. This paper presents a dedicated modeling language showing firstly how it is possible to ease the modeling process and secondly how the semantic gap between modeling logic and the domain can be reduced.

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 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 the one hand, a dedicated language does not enable per se the solution of the models it defines. On the other hand, 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. Firstly, this is caused by the number of configurations of variables of the model, if based on state space oriented analytical techniques. Secondly, the solution may require a long and complex specification of all required elements of the system, if based on generic architectural simulation approaches.

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 literature offers appropriate frameworks to support language development and solution process management. A proposal for a solution technique based on the same approach is currently being submitted for publication.

The paper is organized as follows: the first Section presents related works; the next Section is dedicated to the description of the modeling language, this is followed by a case study; the final section provides the conclusions and future work.

RELATED WORK

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 Manyika et al. (2011); Eco (2011)), as well as research and academia. Big Data poses important research challenges, with reference to functional and non functional specifications on data and processes.

Proceedings 27th European Conference on Modelling and Simulation ©ECMS Webjørn Rekdalsbakken, Robin T. Bye, Houxiang Zhang (Editors)
ISBN: 978-0-9564944-6-7 / ISBN: 978-0-9564944-7-4 (CD)

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 (Wu et al. (2012); Madden (2012)), and a good presentation of the main themes is given in (Bertino et al. (2011); Fu et al. (2012); Bryant et al. (2008); IBM et al. (2011)). From the architectural point of view Apache Hadoop (Had (2008); White (2009)) 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 Isard et al. (2007b) 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 (?) 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 (?) and Apache Cassandra (?) address the issue raised by relational databases regarding scalability, high availability and performance.

The literature presents some relevant contributions to solve part of the architectural problems. In (Tierney et al. (2012)) 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 (Zahavi et al. (2012)) 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 (Fu et al. (2012)), to support compliance to QoS parameters. Finally, (Jung et al. (2012)) 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 (Dai et al. (2011)), 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 supporting design. A sophisticated synthetic workload generator for map-reduce applications over cloud architectures is described in (Chen et al. (2010)), which is used to evaluate performance trade-offs. It can be a significant support tool for other modeling methods. In (Shi et al. (2010)) a system for benchmarking cloud-based data management systems is presented, giving useful hints on storage performance in the most popular Big Data environments.

FRAMEWORKS FOR THE DESIGN OF MODELING LANGUAGES

The main approach in literature for the definition of custom models and modeling formalisms is based on metamodeling (Bézivin (2005); Van Gigch (1991); Jeusfeld et al. (2009)), a consolidated conceptual tool that has been successfully exploited in several cases (e.g. Model Driven Engineering (Poole (2001)), software engineering (Group (2010)) and multiformalism modeling (Vittorini et al. (2004); Gholizadeh and Azgomi (2010); de Lara and Vangheluwe (2002); Iacono et al. (2012))).

One of the most widespread metamodeling approaches is given by the eCore (Steinberg et al. (2008)) 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 that describe its business logic, its architecture and the relations between the objects that form 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 literature by two different frameworks, OsMoSys and SIMTHESys. Both support the implementation of multiformalism performance modeling techniques. The OsMoSys framework (Gribaudo et al. (2003); Vittorini et al. (2004); Moscato et al. (2007); Franceschinis et al. (2004); Vittorini et al. (2002); Franceschinis et al. (2002a,b, 2009)) 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 (Barbierato et al. (2011a,b,c); Iacono and Gribaudo (2010); Iacono et al. (2012); Barbierato et al. (2012a,b)) aims to provide a tool for the rapid development of new formalisms and the automatic generation of related (multiformalism) solvers. Similarly 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) that have 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 that use 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.

THE 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 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 that form 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. The operational elements model the sequence of operations in a map-reduce pipeline and consist of i) *Trigger* representing a data source that generates 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.

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 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 represents computing nodes that exploit 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. A *Trigger* element is characterized by the *Rate* property, representing the arrival rate of requests. To simplify the computation of performance indices, we consider Poisson arrival processes. A Map element is characterized by the *MapEffort* property that describes the complexity of the map operation. In particular, we imagine that 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 that describes the time required to perform a reduction. This time can be composed by 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 *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, represent 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 De-

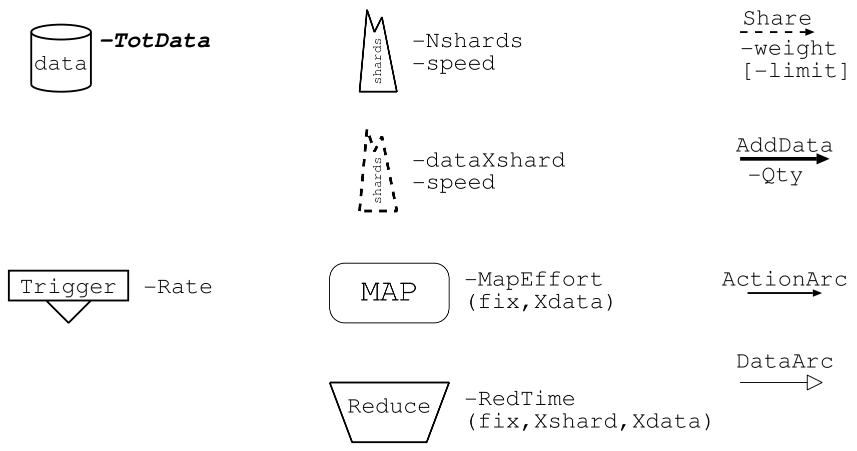

Fig. 1. Elements of the Big Data formalism

scription 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.

A CASE STUDY

The case evaluated in this paper describes a real system, of which 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 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 the Art *DataSet* element, on which it operates, through which 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 that performs 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, we imagine 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 we imagine 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.

We begin our study by fixing the number of article $N = 10000$, 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, we have a large response time due to the high work that the few shards have to perform

[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

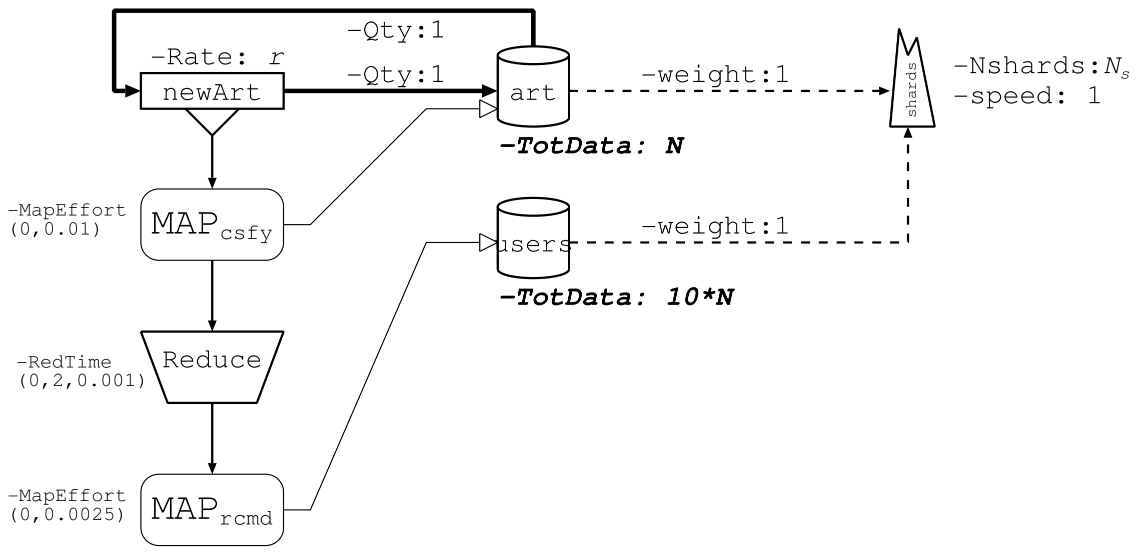

Fig. 2. Model of the case study

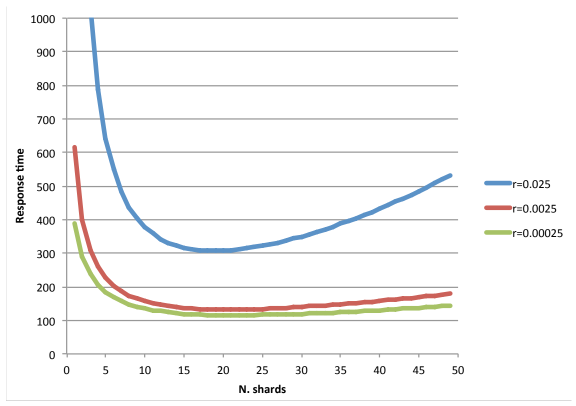

Fig. 3. Response time for a varying number of shards for different article arrival rates r.

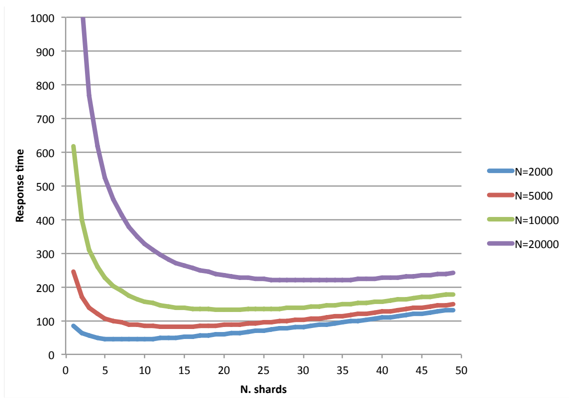

Fig. 4. Response time for a varying number of shards for different number of articles N.

to complete the map operations. When however 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, makes 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, we can see that the number of shards that gives the minimum response time ($N_s = 22$) is independent on the article arrival rate.

We then study the effect of changing the number of articles N (and thus proportionally the number of users), with a fixed arrival rate of $r = 0.0025\ art/sec$. 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 an 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 Fig. 3 and Fig. 4 to simplify the presentation. The obtained intervals were very tight, as can be seen in Table I for some number of shards N_s, considering an arrival rate of new articles $r = 0.0025\ art./min$, and a dataset size of $N = 10000$.

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

TABLE I: 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 $N = 10000$.

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 was to allow Big Data application designers and Big Data system administrators to evaluate their choices and experiment with what-if analysis. From the authors' evaluation, the proposed language 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.

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 user intervention on the model.

References

(2007). Microsoft Dryad.

(2008). Apache Hadoop.

(2011). Drowning in numbers – digital data will flood the planet—and help us understand it better. *The Economist*.

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

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

Barbierato, E., Gribaudo, M., and Iacono, M. (2011b). 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*.

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

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

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

Bézivin, J. (2005). On the unification power of models. *Software and System Modeling*, 4(2):171–188.

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

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

Dai, J., Huang, J., Huang, S., Huang, B., and Liu, Y. (2011). 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. USENIX Association.

de Lara, J. and Vangheluwe, H. (2002). Atom3: A tool for multi-formalism and meta-modelling. In Kutsche, R.-D. and Weber, H., editors, *FASE*, volume 2306 of *Lecture Notes in Computer Science*, pages 174–188. Springer.

Franceschinis, F., Gribaudo, M., Iacono, M., Mazzocca, N., and Vittorini, V. (2002a). Towards an Object Based Multi-Formalism Multi-Solution Modeling Approach. In *Proc. of the Second International Workshop on Modelling of Objects, Components, and Agents (MOCA'02), Aarhus, Denmark, August 26-27, 2002 / Daniel Moldt (Ed.)*, pages 47–66. Technical Report DAIMI PB-561.

Franceschinis, G., Gribaudo, M., Iacono, M., Marrone, S., Mazzocca, N., and Vittorini, V. (2004). Compositional modeling of complex systems: Contact center scenarios in OsMoSys. In *ICATPN'04*, pages 177–196.

Franceschinis, G., Gribaudo, M., Iacono, M., Marrone, S., Moscato, F., and Vittorini, V. (2009). 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. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

Franceschinis, G., Gribaudo, M., Iacono, M., Vittorini, V., and Bertoncello, C. (2002b). DrawNet++: a flexible framework for building dependability models. In *In Proc. of the Int. Conf. on Dependable Systems and Networks*, Washington DC, USA.

Fu, Y., Jiang, H., and Xiao, N. (2012). 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. Springer-Verlag New York, Inc.

Gholizadeh, H. M. and Azgomi, M. A. (2010). A meta-model based approach for definition of a multi-formalism modeling framework. *International Journal of Computer Theory and Engineering*, 2(1):87–95.

Gribaudo, G., Iacono, M., Mazzocca, M., and Vittorini, V. (2003). The OsMoSys/DrawNET Xe! Languages System: A Novel Infrastructure for Multi-Formalism Object-Oriented Modelling. In *ESS 2003: 15th European Simulation Symposium And Exhibition*.

Group, O. M. (2010). Unified modeling language standards version 2.3.

Iacono, M., Barbierato, E., and Gribaudo, M. (2012). The SIMTHESys multiformalism modeling framework. *Computers and Mathematics with Applications*.

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

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

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

Isard, M., Budiu, M., Yu, Y., Birrell, A., and Fetterly, D. (2007b). 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, pages 59–72, New York, NY, USA. ACM.

Jeusfeld, M. A., Jarke, M., and Mylopoulos, J., editors (2009). *Metamodeling for Method Engineering*. MIT Press, Cambridge, MA, USA.

Jung, G., Gnanasambandam, N., and Mukherjee, T. (2012). 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, pages 811–818, Washington, DC, USA. IEEE Computer Society.

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

Manyika, J., Chui, M., Brown, B., Bughin, J., Dobbs, R., Roxburgh, C., and Byers, A. H. (2011). Big Data: The next frontier for innovation, competition, and productivity. *Report McKinsey Global Institute*.

Moscato, F., Flammini, F., Lorenzo, G. D., Vittorini, V., Marrone, S., and Iacono, M. (2007). The software architecture of the OsMoSys multisolution framework. In *ValueTools '07: Proceedings of the 2nd international conference on Performance evaluation methodologies and tools*, pages 1–10.

Poole, J. D. (2001). *Model-Driven Architecture: Vision, Standards, And Emerging Technologies. Position Paper Submitted to ECOOP 2001*.

Shi, Y., Meng, X., Zhao, J., Hu, X., Liu, B., and Wang, H. (2010). 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. ACM.

Steinberg, D., Budinsky, F., Paternostro, M., and Merks, E. (2008). *EMF: Eclipse Modeling Framework, 2nd Edition*. Addison-Wesley Professional.

Tierney, B., Kissel, E., Swany, D. M., and Pouyoul, E. (2012). Efficient data transfer protocols for Big Data. In *eScience*, pages 1–9. IEEE Computer Society.

Van Gigch, J. P. (1991). *System design modeling and metamodeling / John P. van Gigch*. Plenum Press, New York :.

Vittorini, V., Franceschinis, G., Gribaudo, M., Iacono, M., and Mazzocca, N. (2002). 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.

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

White, T. (2009). *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 1st edition.

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

Zahavi, E., Keslassy, I., and Kolodny, A. (2012). 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, pages 99–110, New York, NY, USA. ACM.

AUTHOR BIOGRAPHIES

ENRICO BARBIERATO is a Consultant working for the IT industry. He earned a BSc (Hon) at University of Turin (Italy), an MSc in Advanced Studies in Artificial Intelligence at the Katholieke Universiteit of Leuven (Belgium) and a PhD in Computer Science at the University of Turin (Italy). His research activity concerns performance evaluation of multiformalism models. His email is

enrico.barbierato@mfn.unipmn.it.

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 spatial models. The main applications to which the previous methodologies are applied comes from cloud computing, multi-core architectures and wireless sensor networks.

MAURO IACONO is a tenured Assistant Professor and Senior Researcher in Computing Systems at Dipartimento di Scienze Politiche, Seconda Università degli Studi di Napoli, Caserta, Italy. He received a Laurea in Ingegneria Informatica (MSc) degree cum laude (Hon) in 1999 by Università degli Studi di Napoli "Federico II", Napoli, Italy, and a Dottorato in Ingegneria Elettronica (PhD) degree by Seconda Università degli Studi di Napoli, Aversa, Italy. He published over 35 peer reviewed scientific papers on 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 http://www.mauroiacono.com.