A Configuration-Independent Score-Based Benchmark for Distributed Databases

Claudio A. Ardagna, Ernesto Damiani, Fulvio Frati, Davide Rebeccani

Abstract—The business potential of big data is leading to a *data-driven economy*, where low-cost and low-latency data analysis represents a major competitive advantage. The research community has proposed many technological solutions for big data, such as NoSQL databases, which are difficult to evaluate and compare via standard IT procurement procedures. In addition, lack of competences in big data domains make procurement of big data solutions a tedious and uncertain process, which might impair the success of a business. In this paper, we present a score-based benchmark for distributed databases, which supports adopters in selecting a solution that fits their needs. The proposed benchmark is independent from the configurations of the specific database and deployment environment, requires low effort on the part of end users, is extensible and can be applied to both SQL and NoSQL databases, can be used to evaluate databases according to different properties (e.g., performance, consistency), and can be integrated with existing benchmarks to reduce the burden of their execution. We experimentally evaluate our methodology to validate its effectiveness.

Index Terms—Big Data, Cloud, NoSQL Database, Score-Based Benchmark.

1 Introduction

We live in a pervasive and interconnected world, where users are surrounded by sensors and continuously interact with external actors, almost anywhere and anytime. A huge amount of data are generated and collected every minute – 1.7 million billion bytes of data, that is, over 6 megabytes for each human being on the planet Earth according to [1] – from a wide variety of sources (e.g., sensors, mobile devices, social media). Low latency access to huge distributed data sources has become a value proposition for the next generation of business intelligence applications. Unfortunately technologies for big data analysis and management are often difficult to evaluate using standard IT procurement procedures and difficult to compare with traditional techniques. Being able to select the technology and storage infrastructure that best fit a user's requirements has become a key competitive advantage in data-driven economy, and sparked the interest of the research community in the definition of benchmarks that comparatively evaluate existing (big) data solutions.

Current and past research has defined several different benchmarks to compare IT systems (including traditional databases) [2], [3], [4], [5], [6], [7]. With the advent of the cloud, new benchmarks have been proposed [8], [9], [10], [11], evaluating elasticity and scalability solutions, hardware and software performance, and virtualization technologies. Approaches specifically designed for big data infrastructures have been proposed in the last years and focused on NoSQL databases [12], [13], [14]. At the same time, benchmarks for NoSQL databases emerged (e.g., [15], [16], [17], [18], [19]). They mostly aimed to provide a solution for performance evaluation that allows users to compare different solutions for big data management. However, these proposals do not generalize well, since they focus on specific environments and configurations, and require a substantial effort on the part of adopters to evaluate databases in their own infrastructure. Also, according to [19], the relative youth of big data systems introduces the risks of *benchmarketing*, where vendors define benchmarks to maximize their product evaluation, or *benchmark-specials* attitude, where vendors configure their products to retrieve the best evaluation from a selected benchmark. Finally, current approaches do not provide support for comparing SQL and NoSQL databases, which in specific scenarios might show comparable behaviors.

According to Jim Gray's seminal work on database benchmarks [20], a benchmark must be *relevant*, *portable*, *scalable*, and *simple*. Also, diversity of computer systems calls for domain-specific benchmarks that define synthetic workloads characterizing typical applications. With the advent of the cloud and the additional diversity introduced by NoSQL databases, the need of a simple and configuration-independent benchmark for distributed databases is even exacerbated. However, the complexity of the underlying technologies and the impact the available infrastructures have on performance make the application of traditional benchmarks either difficult or ineffective. In fact, on one hand, the deployment of several distributed databases requires competences which are not common even in database experts; on the other hand, performance measurements done with configurations of the database and its deployment environment different from the ones used in the target scenario can highly affect the precision of retrieved results.

In this paper, we define a generic, configuration-independent, extensible, and low-overhead score-based

• *Claudio A. Ardagna, Ernesto Damiani, Fulvio Frati, and Davide Rebeccani are with Dipartimento di Informatica, Università degli Studi di Milano, 26013 Crema – Italy. Ernesto Damiani is also with Etisalat British Telecom Innovation Center, Khalifa University of Science, Technology and Research, Abu Dhabi, UAE.*
E-mail: firstname.lastname@unimi.it

benchmark. Our design departs from the practice of developing benchmarks that apply in specific scenarios and to specific configurations only. We first describe a taxonomy for SQL and NoSQL databases that is at the basis of our score-based benchmark. We then define a methodology based on users' preferences in profiles, which allows to evaluate a database in a specific deployment environment, by simply specifying the database configuration, the expected workload, and the property to be evaluated (e.g., performance, consistency). The proposed approach is based on a weighted sum of elements in the taxonomy, which are relevant for the considered scenarios. Suitable weights are generated by empirical testing and regression analysis, making our methodology flexible, extensible, and easily manageable by a community. We note that, while the execution of testing and regression analysis requires competences and introduces overheads comparable to the ones of current benchmarks, weight selection is done once for a specific deployment environment and configuration, and then reused for similar scenarios. The availability of weights allows non-expert users to only define the configurations of their system and run the benchmark, with no need of multiple system deployment and evaluation.

Being configuration independent, our score-based benchmark can be applied alone or in conjunction with existing benchmarks, and compare both SQL and NoSQL databases. It can then be used either to select the best database or to narrow the search space of existing benchmarks on the basis of user's preferences.[1] In the latter case, a user executes the methodology in this paper to identify the subset of n best databases, and then applies an existing benchmark to select the best one in the subset. Our benchmark supports comparison based on traditional performance property, as well as on additional properties such as availability and consistency.

The contribution of this paper is threefold: *i)* we define a taxonomy of parameters that characterize SQL/NoSQL databases and could influence their behavior; *ii)* starting from the proposed taxonomy, we define a score-based benchmark that applies a weighted sum of taxonomy parameters, where weights are calculated according to linear regression analysis; *iii)* we experimentally evaluate the precision of the proposed solution, including the approach for weight selection.

The remainder of this paper is organized as follows. Section 2 presents related work. Section 3 describes our taxonomy of database parameters. Section 4 illustrates the proposed score-based benchmark, also describing the instantiation of relevant parameters and the approach to weight selection. Section 5 presents the experimental evaluation of our methodology. Section 6 discusses pros and cons of our approach. Section 7 draws our conclusions.

2 Related Work

Benchmarks have been defined since the advent of IT and targeted several different and heterogeneous aspects of past and current digital systems (e.g. [2], [4], [5], [6], [7]). Today, the research community has given a lot of attention to the problem of benchmarking the cloud in all its facets. Several approaches have been proposed ranging from the evaluation of (distributed) databases, to the evaluation of cloud management approaches.

The line of research closest to ours concerns the definition of benchmarks for NoSQL databases. The problem of benchmarking NoSQL databases is particularly relevant as well as challenging due to the heterogeneity of the provided NoSQL approaches, which can vary depending on many parameters such as supported data model (e.g., key value, document, graph, column) [12], [13], [14]. Armstrong et al. [15] describe LinkBench, a synthetic benchmark for databases based on streams of data from Facebook social graphs. The authors model query and data workload in many dimensions, which are used for testing social data storage. Chen et al. [21] provide a suite of workloads for the evaluation of MapReduce performance. The proposed solution studies two MapReduce traces and provides a vocabulary for the description of relevant workloads. It also proves how the use of realistic workloads can increase the quality of the evaluation and of the MapReduce task scheduler choice. Dory et al. [17] present some results on the scalability and elasticity of Cassandra, HBase, and MongoDB, under realistic loads using the Wikipedia database. Dede et al. [18] experimentally evaluate the performance of MongoDB and Hadoop for scientific data analysis. In particular, the paper presents a deep evaluation of performance, scalability, and fault tolerance when MongoDB is integrated with Hadoop. A comparative evaluation between MongoDB and HDFS is also provided. Ghazal et al. [19] describe BigBench an end-to-end big data benchmark for industry and big data analytics. It provides a data model and a generator that accomplish the three *V* of big data systems, namely *Variety*, *Velocity*, and *Volume* [22]. It supports structured, semi-structured, and unstructured data. Similar to our approach, the proposed workload is built on a set of queries against a given data model. We note that, in a recent solution [23], big data are defined using the 5V model: Volume, Variety, Velocity, Value, Veracity. Cooper et al. [16] present the *Yahoo! Cloud Serving Benchmark* (YCSB) framework. YCSB is a benchmark for comparing the performance of NoSQL databases. It defines several workloads, query types, and query distributions, and uses them for evaluating performance of several databases including Cassandra, HBase, PNUTS, and MySQL. Li and Manoharan [24] present a comparative evaluation of SQL and NoSQL databases on key-value stores. To the best of our knowledge, differently from the solution in this paper, existing benchmarks *i)* do not support comparison of SQL and NoSQL databases (only [25] compares the popularity of SQL and NoSQL database in a single ranking), *ii)* require high competences and put high overhead on final users, *iii)* require experimental evaluation of each candidate database, *iv)* consider some specific scenarios in terms of workload, deployment environment, therefore being the results of the benchmark affected by changes in the real user scenario, *v)* are architecture dependent.

Other works discussed specific benchmarks for different aspects of the cloud. Li et al. [26] describe a set of metrics

1. We note that, often, expert users manually reduce the search space on the basis of their experience. This is however not always possible and successful, especially in cases of conflicting requirements or non-expert users.

and benchmarks that can be used to evaluate commercial cloud services. Islam et al. [9] present a benchmark based on a penalty-model for evaluation of elasticity effectiveness. Much in line with the work in this paper, the proposed benchmark summarizes penalty rates in a single score for comparison. However, differently from our approach, the score is a normalization on a reference system and is calculated through real measurements on the system under evaluation. Turner et al. [11] describe C-Mart, a benchmark for the cloud that emulates the behavior of a modern application in the cloud. It provides a web application, a client emulator, a deployment server, and a scaling API to produce a testing environment that is faster and more realistic than existing benchmarks. It evaluates management systems, application scaling, VM consolidation, and performance. Rabl et al. [10] propose a generator of data for cloud-scale benchmarking. The proposed data generator considers cloud peculiarities and is therefore designed for cloud evaluation. Ferdman et al. [8] present the cloudsuite benchmark for emerging scale-out workloads. The benchmark is aimed at evaluating the performance of modern and predominant CPU micro-architectures, which are found to be inefficient for the proposed cloud-oriented workloads.

Most of existing benchmarks are highly dependent on the execution environment and on users requirements. Users need to execute benchmarks on premises to find out the best solution that fits their requirements. To address this shortcoming, Liquid Benchmarking [27] proposes a benchmarking-as-a-service. The Liquid Benchmark platform implements a SaaS application supporting the execution of independent experimental evaluation and comparison techniques to evaluate algorithms, solutions, and systems in a consistent way. It supports the definition of standard benchmarks following a collaborative effort and repeatability of results. Still, there is no possibility of benchmarking a given algorithm or solution according to a specific deployment environment. Also, Liquid Benchmarking focuses on algorithms and solutions as services rather than data-driven infrastructures like NoSQL databases.

Some approaches focused on the evaluation of the performance of cloud scalability and elasticity algorithms [28], [29]. For instance, Iosup et al. [30] analyze the performance of cloud computing services for scientific computing workloads, while Salah and Boutaba [31] define a model for evaluating elastic cloud applications by estimating service response time. Stantchev [32] describes a methodology to performance evaluation of cloud computing solutions varying the platform instances from one to three. Espadas et al. [33] provide a formal measurement for under and over provisioning of cloud resources, and an approach based on resource allocation for SaaS to create a cost-effective scalable environment. Copil et al. [34] present a multi-layer approach for the management of elasticity and scalability in the cloud. Other works have focused on the flexibility of NoSQL databases and storage, analyzing the process of adaptive expansion and contraction of resources, and providing techniques for performance evaluation and control [35], [36], [37].

Finally, a variety of issues have been studied in the context of performance optimization in the cloud. Some approaches have focused on the problem of resource and data allocation [38], [39]. Many researchers undertook the task of defining metrics and benchmarks to evaluate the ability of a cloud framework to adapt to variable loads [9], [40], [41]. Ali-Eldin et al. [42] also present an adaptive controller for cloud infrastructures supporting proactive and reactive horizontal elasticity. Fito et al. [43] present an elastic cloud hosting provider, building on service-level agreements, to maximize provider revenue. Klems et al. [44] present a load balancer for the Sherpa storage system, which automatically balances the load by moving tablets from overloaded servers or by splitting data in different tablets. Curino et al. [45] introduce Schism, a scalability solution for shared-nothing distributed databases, which is based on workload-aware database partitioning and replication.

3 A Taxonomy of Parameters Influencing Database Behavior

We describe a taxonomy of parameters influencing database behavior, which is adopted as the basis for the definition of our score-based benchmark [46]. We first introduce the basic concepts that are used to design our taxonomy (Section 3.1), and, then, present its building blocks (Section 3.2). Concepts and building blocks are general enough to embrace the entire set of distributed databases, which are heterogeneous by nature.

3.1 Basic Concepts

Our taxonomy relies on three basic concepts, namely, *macro-area*, *attribute*, *parameter*, which map on three levels of the taxonomy tree. It is built around the concept of *attribute*, that is, a combination of parameters modeling a specific characteristic of a database and its environment. In particular, an attribute a represents a specific characteristic of a database and its environment, and is defined as a set $\{p_1,\ldots,p_n\}$ of parameters, representing different implementations/configurations of the same attribute. As an example, attribute *scalability* can refer to the implemented scalability algorithm, *data model* to the adopted data model, and *persistency* to the deployed solution providing data persistency.

Parameters define possible architectural choices done in the configuration of a database for supporting a specific attribute. Each parameter is a measurable artifact that represents an approach to support a specific attribute and is defined as a set $\{l_1,\ldots,l_n\}$ of levels. Levels categorize all admissible implementations, and are defined as a pair $(name,val)$, where $l_i.name$ is the name of the i-th level and $l_i.val$ its score.

A total order $\preceq_p$ can be straightforwardly defined over levels based on scores and is such that for each pair of levels l_i and l_j of a given parameter p, $l_i.name \preceq_p l_j.name$ iff $l_i.val \leq l_j.val$. Each parameter can be *platform-dependent* or *platform-independent*, and is initialized with the value of the selected level. A platform-dependent parameter shows a dependence on other database attributes/parameters and/or the environment where the database is deployed; a platform-independent one is generic for all databases and environments. For instance, the parameter *Key-value* of the attribute *Data Model* is platform-independent; all parameters of attribute *Storage* are platform-dependent.

All attributes are finally collected in macro-areas, which can be extended and refined as needed, and refer to specific aspects of a given database implementation. A *Macro-area* is defined as a set $\{a_1,\ldots,a_n\}$ of specific attributes.

An example of macro-area is *DB-Topology*, which refers to all attributes concerning the database structure.

3.2 Building blocks

We describe our taxonomy according to three main macro-areas (see Figure 1): *DB Topology*, *Query Type*, and *System and Network Topology*. For each of them, we discuss its rationale, and corresponding attributes and parameters. We note that our taxonomy is not meant to be exhaustive, can be extended according to the considered scenario without affecting the proposed benchmark and methodology (Section 4), and can also be used for SQL databases. The taxonomy is the core of our approach since it specifies the attributes and parameters which are evaluated by our benchmark.

3.2.1 Query Type

Macro-area *Query Type* collects all artifacts related to the way in which a given DB manages and supports the execution of given classes of queries, and how the DB supplies services to external clients. In other words, it evaluates the effectiveness of the database in executing specific classes of queries of interest for the user. In case no preferences are specified, a complete evaluation is provided independently by the specific query type. Box with solid line in Figure 1 shows the part of our taxonomy regarding macro-area query type.

This macro-area is composed of two distinct attributes. *Query API Model* contains parameters like Map&Reduce, Graph, Collection, and describes how a DB provides query services to external clients *Query Model* includes parameters that describe the methods used to implement and execute queries.

3.2.2 DB Topology

Macro-area *DB Topology* collects all artifacts that refer to the DB structure. It evaluates all technologies that contribute to the supply of DB functionalities and how much these functionalities fit the requirements specified by the user. Box with dashed line in Figure 1 shows the part of our taxonomy regarding macro-area DB topology, which includes the following attributes:

- *Scalability*: it describes the technologies used by the DB to scale under heavy workloads.
- *Data Model*: it describes how data are organized in the DB.
- *Persistency*: it describes how data writing and reading are managed by the DB. It also specifies how the system maintains a consistent state after the execution of operations.
- *Versioning*: it describes alternative technologies that can be used to manage different versions of data. It specifically focuses on the scenario where the DB needs to manage concurrent data writing guaranteeing the consistency of data.
- *Partitioning*: it describes the technologies used by the system to replicate and partition data within different nodes, ensuring availability, reliability, and consistency of the overall framework.
- *Storage Layout*: it describes how the storage can be accessed and its influence on performance. It specifies which types of data can be read in block.

3.2.3 System and Network Topology

Macro-area *System and Network Topology* collects all artifacts that refer to the system structure, to the characteristics of the environment where the database is deployed, and to the network structure used by database components to communicate and exchange data. It evaluates the hardware, software, and network configurations to measure the effects the system and network have on the overall database performance. Box with dotted line in Figure 1 shows the part of our taxonomy regarding this macro-area. It contains, on one side, three main attributes referring to CPU, memory, and storage of the system, and, on the other side, an attribute describing the network technology.

The parameters in this macro-area are defined as a function $f(x)$ over the corresponding attribute x. Function $f(x)$ is computed according to the configuration of parameters in other macro-areas and of the environment where the database is deployed. As a consequence, only a single parameter for each attribute can assume a value different from zero for a given database under evaluation. Each parameter value (i.e., the value of function $f(x)$) is produced by experimentally evaluating a representative DB for the specified configuration *conf*(Q,*DB*,*SN*), where Q refers to parameters in macro-area Query Type, *DB* refers to parameters in macro-area DB Topology, and *SN* refers to the deployment environment. As an example, let us consider parameter "$f(M)$ with $conf_i(Q_i,DB_i,SN_i)$" of attribute *Memory*, which evaluates the impact of the memory on a given database behavior. For example, relevant configuration $conf_i$ can consider parameters *Sharding* and *Map&Reduce*, and deployment configurations *CPU* and *Storage*. A function $f(M)$, modeling the performance of a representative database implementing a sharding approach over map&reduce query API model, with fixed CPU and storage, and varying the amount of main memory, is produced and used to assign values to parameter "$f(M)$ with $conf_i(Q_i,DB_i,SN_i)$" of attribute *Memory*.

As a concluding remark, we note that the taxonomy in this section includes parameters that might affect a database behavior. Though our taxonomy has been designed to be as inclusive as possible and to target both SQL and NoSQL databases, new macro-areas, attributes, and parameters can be added when needed or it can be extended on the basis of existing taxonomies (e.g., [47], [48]). This approach allows our solution to promptly react to changes in database implementation, which are quite common in current scenarios, with minimum impact on the methodology discussed in Section 4.

4 Score-based benchmark

Our score-based benchmark supports a reference scenario that involves three main parties: *i)* a *database provider*, implementing the database target of our benchmark-based

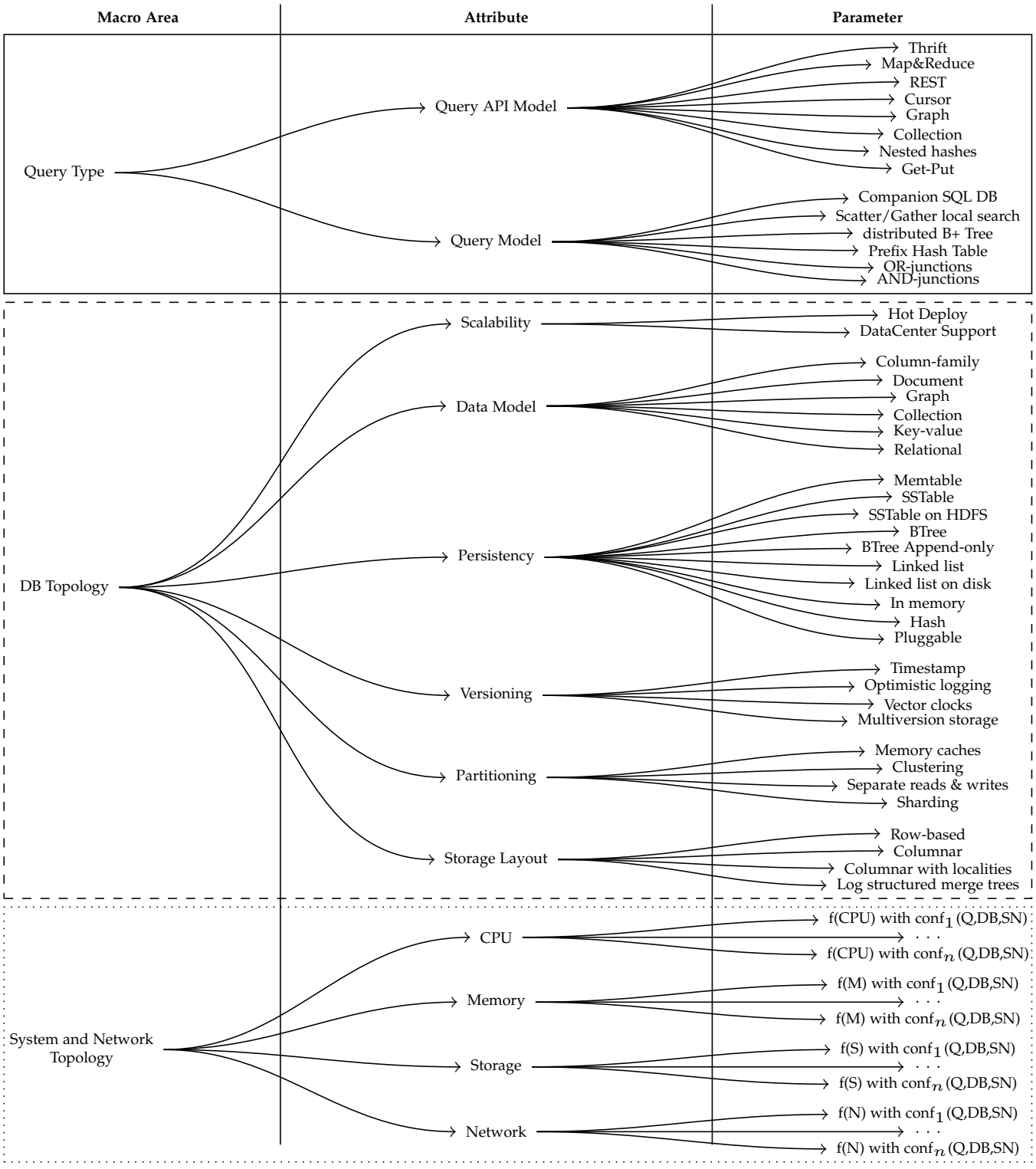


Fig. 1. Taxonomy

evaluation; *ii)* an *independent database expert*, conducting experimental evaluation at the basis of our score-based benchmark; *iii)* a *user*, looking for a database and using our benchmark to evaluate who is the best one that fits its preferences in an optimal way. Our benchmark implements a methodology for evaluating databases in a specific configuration, independently from the database providers and according to preferences of the users. Preferences are specified in a profile $\Pi_{pr,wl}$ that can be defined as a pair (*pr*,*wl*), where pr is the property to be evaluated (e.g., *performance*, *availability*) and wl is the workload that the database under evaluation is expected to manage (e.g., *write-only*, *read-only*, *mostly-write*, *mostly-read*) [49]. $\Pi_{pr,wl}$ drives the combination of parameters to calculate a database score.

Overall, our benchmark permits comparison of databases, without the need for users to execute complex testing activities, claiming which database is more suitable in specific database and environment configurations.[2] To this aim, the evaluation process proceeds as follows. First, the user defines its profile on the basis of the property to be verified and the expected workload. The profile permits to select a collection of weights, which have been previously calculated according to experimental activities done by experts and are used to integrate parameter values in a single database score. Upon a specific database is selected for evaluation, each platform-independent parameter is instantiated by the user with the value of the level supported by the database configuration. Similarly, based on the selected database and configuration of deployment environment, each platform-dependent parameter is valued. Levels of independent and dependent parameters are specified in different ways as discussed in Section 4.1. As soon as each parameter has a value, a weighted sum of parameters is calculated (Section 4.3) on the basis of weights selected using the profile and defined in Section 4.2. We note that the profile could be extended to include low-level details on the physical deployment environment (e.g., characteristics of CPU, memory, and storage chips and devices), improving the quality of selected weights. For simplicity, in the following, we consider a profile composed of workload and property only.

In the remainder of this section, we first present two important aspects at the basis of our benchmark specification, namely, level definition (Section 4.1) and weight selection (Section 4.2), and then formalize the benchmark itself (Section 4.3).

4.1 Parameter level definition

An important aspect of our benchmark is the definition of the levels and corresponding values for each of the platform-independent (Section 4.1.1) and platform-dependent (Section 4.1.2) parameters in Figure 1.

4.1.1 Platform-independent parameters

They are defined as a set of discrete levels, each one representing a possible implementation for the parameter (i.e., a possible instantiation of the functionality related to the parameter). Ordering between levels is provided according to the profile. The default assignment function gives proportional values in (0,1] to the levels based on their ranking. Assuming n levels for parameter p, $l_i.val{=}i/n$, with i=1,...,n. As an example, three levels are defined for parameter *Sharding*: *Not-Supported*, *Optional*, and *Native*. Assuming property performance is specified in the profile, each level is associated, in crescent order of importance and guaranteed performance, with values $1/3$, $2/3$, and 1, respectively.

We note that some platform-independent parameters can assume two possible values only: *1* meaning that the database infrastructure *supports* the parameter; $1/2$ meaning that the parameter is *unsupported* (i.e., the database does not provide the related functionality). The parameters associated with attribute *Data Model* are clear examples of binary level parameters. In addition, the support of data models is a strategic feature of each DB engine, and only a combination of models can be adopted at a time. If we take MongoDB as an example, parameter *document* and *collection* are valued with 1, all other parameters have value 0.

Variations to the default assignment function ($l_i.val{=}i/n$) can be provided by expert users on the basis of the specific scenario. For instance, an ad hoc value is usually provided for the lowest level (e.g., *not supported*) of each parameter.

Example 4.1. Let us consider a profile with property *performance* and a *mostly-read* workload (95% read, 5% update), representing a typical distributed model. We select a subset of the platform-independent parameters in our taxonomy in Figure 1, which are likely to impact the performance of a database, and define levels (i.e., level names and values) for them. Among the independent parameters in macro-areas *Query Topology* and *DB Topology* in Figure 1, we select the following ones: *i) Thrift*, *Map&Reduce*, *REST*, *Cursor*, *Collection*, they influence the management of data and, hence, the performance of databases, especially for high request rates; *ii) Companion SQL DB*, *Distributed B+ Tree*, *Prefix Hash Table*, *OR-junctions*, *AND-junctions*, they permit to distinguish database performance on the basis of the supported query model; *iii) Hot Deploy and DataCenter Support*, they permit to analyze the impact of scalability features on database performance; *iv) Column-family*, *Document*, *Graph*, *Collection*, *Key-value*, *Relational*, they analyze the impact of the supported data model on database performance; *v) Memtable*, *BTree*, *Hash*, they represent approaches to persistency that could increase the availability and the speed in data collection; *vi) Memory Caches*, *Clustering*, *Separate Reads&Writes*, *Sharding*, they are likely to influence database performance especially in case of distributed queries. Table 1 describes the parameter levels, and their ordering and value depending on the selected profile. We note that we considered a variation to the default assignment function (i.e., $l_i.val{=}i/n$) by assigning value $l_1.val{=}1/10$ for each parameter.

4.1.2 Platform-dependent parameters

They are defined as a set of continuous levels, represented as a function $f(x)$. Variable x represents a measurable artifact that is used to assign a value to the parameter. Function $f(x)$ can be estimated by means of experimental evaluation. For instance, as discussed in Section 3.2.3, attribute *Memory* is composed of n platform-dependent parameters "$f(M)$ with $conf_i(Q_i,DB_i,SN_i)$". Relevant configuration $conf_i$ considers a combination of parameters in macro-area Query Type (Q_i), macro-area DB Topology (DB_i), and macro-area System and Network Topology (SN_i). A function $f(M)$ is then produced according to $conf_i(Q_i,DB_i,SN_i)$ and used to calculate the value $f(M)$ of the corresponding parameter. Examples of functions $f(x)$ for platform-dependent parameters are provided in our experimental evaluation in Section 5.2.

2. We note that our benchmark can also be used to compare different databases *a priori*, that is, independently of a specific configuration of the environment, though such comparison is out of the scope of this paper. In this case, platform-dependent parameters are not considered and the benchmark will claim which database is better for the considered property if configured in the optimal way.

TABLE 1
Platform-independent parameter values

	p	$l.name$	$order$	$l.val$
p_1	Thrift	Not-Supported	1	1/10
		Supported	2	1
p_2	Map&Reduce	Not-Supported	1	1/10
		Traditional MR	2	2/3
		YARN	3	1
p_3	REST	Not-Supported	1	1/10
		Custom REST Verbs	2	2/3
		Standard HTTP REST Verbs	3	1
p_4	Cursor	Not-Supported	1	1/10
		Supported	2	1
p_5	Collection	Not-Supported	1	1/10
		Supported	2	1
p_6	Companion SQL DB	Not-Supported	1	1/10
		Standard SQL	2	2/3
		Extendable SQL	3	1
p_7	Distributed B+ tree	Not-Supported	1	1/10
		Supported	2	1
p_8	Prefix Hash Table	Not-Supported	1	1/10
		Supported	2	1
p_9	OR-Junctions	Not-Supported	1	1/10
		Supported	2	1
p_{10}	AND-Junctions	Not-Supported	1	1/10
		Supported	2	1
p_{11}	Hot Deploy	Not-Supported	1	1/10
		Configurable	2	2/3
		Automatic	3	1
p_{12}	Data Center Support	Not-Supported	1	1/10
		Supported	2	1
p_{13}	Column Family	Not-Supported	1	1/10
		Supported	2	1
p_{14}	Document	Not-Supported	1	1/10
		Supported	2	1
p_{15}	Graph	Not-Supported	1	1/10
		Supported	2	1
p_{16}	Collection	Not-Supported	1	1/10
		Supported	2	1
p_{17}	Key Value	Not-Supported	1	1/10
		Supported	2	1
p_{18}	Relational	Not-Supported	1	1/10
		Supported	2	1
p_{19}	Memtable	Not-Supported	1	1/10
		Supported	2	1
p_{20}	Btree	Not-Supported	1	1/10
		Supported	2	1
p_{21}	Hash	Not-Supported	1	1/10
		Supported	2	1
p_{22}	Memory Caches	Default	1	1/10
		Extended	2	1
p_{23}	Clustering	Not-Supported	1	1/10
		Optional	2	2/3
		Native	3	1
p_{24}	Separate R/W	Not-Supported	1	1/10
		Supported	2	1
p_{25}	Sharding	Not-Supported	1	1/10
		Optional	2	2/3
		Native	3	1

4.2 Weight selection

Recalling that our database score calculation is based on a weighted sum of parameters, we define an approach providing near-optimal weights (one for each parameter) that support the precise calculation of the database ranking. We note that weight calculation is done once for each profile and is managed by independent database experts. Weights can then be re-used by users when needed, minimizing the overheads on users themselves and the complexity of our methodology. We also note that weight calculation is independent by database providers, which are not able to influence their values, and in turn the value of the scoring system in Section 4.3.

In this paper, we use a linear regression analysis with constrained least squares estimation. To this aim, we first produce, by means of experimental evaluation, a data set necessary to adjust the variables of our model function for regression analysis. The evaluation is done on real databases or is based on existing benchmark executions (e.g., [16]) for a given profile, in different deployment environments. It produces a total order D=$\{d_1,\ldots,d_t\}$, with $d_i > d_{i+1}$ and i=1,...,$t-1$, where d_i represents a database within a specific deployment environment. Starting from D, we consider all database pairs (d_j,d_k) and generate a data set including data items of the form (x_i,y_i), with i=1,...,m, where $x_i=\{x_{i,1},\ldots,x_{i,n}\}$, with $n<m$, is a set of independent variables and y_i is a dependent variable. More in detail, given a pair (d_j,d_k), where d_j and d_k are either different databases, or the same database with different configurations and/or in different deployment environments, dependent variable y_i comparatively evaluates the experimental results produced by d_j, called y_{d_j}, and the one by d_k, called y_{d_k}. It is calculated as:

$$y_i = \frac{y_{d_j}}{y_{d_k}} \; with \; y_{d_j} \geq y_{d_k}. \tag{1}$$

Each independent variables in $x_i=\{x_{i,1},\ldots,x_{i,n}\}$ refers to one parameter $\{p_1,\ldots,p_n\}$ in Figure 1, and proposes a comparative evaluation between the corresponding parameter of d_j and d_k. Each variable $x_{i,s}$ is calculated as $\frac{p_{j,s}}{p_{k,s}}$.[3] We then propose a linear model where the equation for predicting the i-th data item y_i of the data set assumes the following form:

$$y_i = \sum_{s=1}^{n} x_{i,s} \times w_{p_s} = x_{i,1} \times w_{p_1} + \ldots + x_{i,n} \times w_{p_n}, \tag{2}$$

where each independent variable $x_{i,s}$ is multiplied by a given weight $w_{p_s} \in \mathcal{W}_\mathcal{P}$. The weights are estimated by applying the least squares method, that is, setting each of them to a value minimizing the sum of squared errors according to the data set the model is supposed to fit. In other words, we aim to infer the best mapping that minimizes squared errors, that is, the mismatch between the predicted and expected result.

Formally, the $m>n$ equations of the data set (whose form is presented in Equation 2) can be written as a matrix

$$X\mathcal{W}_\mathcal{P} = Y \tag{3}$$

where

$$\mathrm{X} = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{i,n} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m,1} & x_{m,2} & \cdots & x_{m,n} \end{bmatrix}, \mathrm{W}_\mathcal{P} = \begin{bmatrix} w_{p_1} \\ w_{p_2} \\ \vdots \\ w_{p_n} \end{bmatrix}, Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}.$$

Our goal is to find the vector $\mathcal{W}_\mathcal{P}$ of weights that satisfy the set of equations with minimum error. We use the least squares method that finds the optimum when the sum of squared residuals (i.e., the difference between the value of the dependent variable and the predicted value)

3. We remark that variables are independent by construction, thanks to parameter and level definition in Section 4.1.

$$\mathcal{R}(\mathcal{W}_\mathcal{P}) = \sum_{i=1}^{m} \left| y_i - \sum_{s=1}^{n} X_{is} w_{p_s} \right|^2 = \left\| Y - X\mathcal{W}_\mathcal{P} \right\|^2 \quad (4)$$

is minimum. In other words, the goal is to find the weights $\overline{\mathcal{W}}_\mathcal{P}$ such that

$$\overline{\mathcal{W}}_\mathcal{P} = \arg\min_{\mathcal{W}_\mathcal{P}} \mathcal{R}(\mathcal{W}_\mathcal{P}). \quad (5)$$

We note that not all parameters are relevant for a given profile. Irrelevant parameters p_j must be filtered out a priori or, equivalently, their corresponding weight $w_{p_j} \in \mathcal{W}_\mathcal{P}$ initialized to 0. The linear regression analysis therefore provides support for feature selection, having a predictive value on the abstract model.

Example 4.2. As an example, let us assume a user aiming to select a database with best *performance* (i.e., pr='*performance*') following a typical distributed workload. This scenario considers a workload *mostly-read* (i.e., wl='*mostly-read*') aimed to test databases on their capability of managing distributed computations. Following the specified profile, parameters in attribute *Persistency* will mostly show negative or no impact on property *performance*, while parameters in attribute *Partitioning* will mostly show positive impact on it. The first set of parameters in fact includes aspects of a database that are *likely* to increase data *availability* (i.e., pr='*availability*'), while sacrificing the performance of the database; the second set instead refers to aspects which are *likely* to increase the performance of the database. For instance, let us consider parameter p='*Clustering*' in attribute *Partitioning*. As already discussed, parameter *Clustering* has three levels: *Not-Supported* with value $l_1.val$=1/10, *Optional* with value $l_2.val$=2/3, and *Native* with value $l_3.val$=1. Following our methodology, the weights associated with parameter *Clustering* will change according to the selected profile. For instance, in case pr='*performance*' is selected w_p will assume a positive value; in case either property pr='*availability*' is selected or the workload is such that the overhead given by the clustering process is higher than the real advantages in terms of performance, w_p will assume a lower (if not negative) value.

4.3 Score-based methodology

We now describe how our methodology calculates the score for a database given a specific profile $\Pi_{pr,wl}$. The profile is used to select a collection $\mathrm{W}_\mathcal{P}$ of weights, one for each parameter, as follows.

Definition 4.1 ($W_\mathcal{P}$). Given a property pr and a workload wl specified by the user in a profile $\Pi_{pr,wl}$, a set $\mathrm{W}_\mathcal{P}$ of weights $\{w_{p_1},\ldots,w_{p_n}\}$ is defined/retrieved, such that, for each parameter p_i, there exists a corresponding weight w_{p_i}.

We note that artifacts in the profile are used to index the collection of weights needed to calculate a score for the database. The way in which weights are generated for a given profile is described in Section 4.2.

Upon specifying the profile and selecting the weights, the database under evaluation is analyzed and a single level is selected for each parameter by the user. This means that each parameter p_i is instantiated with the value associated with the selected level l_i (i.e., p_i=$l_i.val$ as discussed in Section 4.1).[4] The database score $\mathcal{S}(\mathrm{W}_\mathcal{P})$ is then calculated according to a specific collection $\mathrm{W}_\mathcal{P}$ of weights as follows.

Definition 4.2 ($S(W_\mathcal{P})$). A database score $\mathcal{S}(\mathrm{W}_\mathcal{P})$ is calculated as

$$\mathcal{S}(\mathcal{W}_\mathcal{P}) = \sum_{\forall p_j} p_j \times w_{p_j} \; with \; w_{p_j} \in \mathcal{W}_\mathcal{P}. \quad (6)$$

In summary, our score-based benchmark implements a provider-independent evaluation methodology. It relies on an empirical set up phase aimed to provide weights, representing a common ground for score calculation and database comparison. This phase substantially reduces the burden on users, which need only to select a profile and specify the configurations (selection of levels in Table 1) of databases target of the evaluation activities. Moreover, the separation between the taxonomy/levels and the score-based methodology makes our approach highly flexible. New macro-areas, attributes, parameters, and levels can be defined without affecting the working of our methodology. In the following of this paper, we discuss the application of our methodology to a real scenario for property performance, showing its effectiveness.

5 Benchmark Evaluation

We experimentally evaluated the effectiveness of our methodology in multiple real scenarios. Here, for space restriction, we report on a single scenario (described in Section 5.1), where we applied our score-based benchmark to a set of different and heterogeneous databases in different configurations and deployment environments. Further experimental results considering a wider variety of databases and configurations are available for interested readers at http://tinyurl.com/onvfg4o. Before evaluating the methodology effectiveness in ranking candidate databases (Section 5.4), we show how relevant parameters and corresponding levels are initialized depending on the considered scenario (Section 5.2), and weights are selected using regression analysis (Section 5.3).

5.1 Experimental settings

We discuss our experimental settings describing hardware and software configurations, target databases, selected profile, and deployment environments.

5.1.1 Hardware configurations, software configurations, and target databases

Our experimental environment consists of a physical machine *Dell PowerEdge 6850* equipped with 4 Intel Xeon Quad Core 2.6 GHz, 16GB ECC RAM, 6x 146 GB Serial Attached SCSI 10K RPM, and 2x 1Gb/s Ethernet NIC. The physical machine hosts XenServer 6.2 deploying different virtual machines over which the databases target of our evaluation have been installed.

4. We note that in case of platform-independent (*a priori*) evaluation, platform-dependent parameters p_i are set to 0.

Target databases have been selected to cover the widest spectrum of database characteristics, the parameters supported in the taxonomy in Figure 1, and the heterogeneity of parameter values. We note that the selected databases have been installed with different configurations and/or within different virtual deployment environments to produce the dataset for tuning our benchmark (training set), as well as the dataset for testing our benchmark (testing set). In particular, we considered the following databases balancing coverage of different database architectures and popularity of selected databases [25]:

- MongoDB v2.4: NoSQL, document-oriented, schema-less database. It is composed of three main components: *i)* shard servers storing data; *ii)* configuration servers storing database configurations; *iii)* a router service receiving and distributing requests to shards.
- Cassandra v2.1.2: NoSQL, column-based, schema-less database. It has a "masterless" architecture, where all peers (nodes) are at the same level. Cassandra supports automatic balance between nodes in the same cluster.
- Hypertable v0.9.8.4 over Hadoop: NoSQL, key-value, schema-less database. It is a massively scalable database representing data as tables of information, with rows and columns. It does not support data types, joins, and transactions.
- PostgreSQL v9.3: SQL, object-relational database system. It is ACID compliant, supports foreign keys, joins, views, triggers, and stored procedures. It supports most SQL:2008 data type and storage of large binary objects. PostgreSQL provides advanced features such as point in time recovery, asynchronous replication, and online/hot backups.
- MariaDB v5.5: SQL, object-relational database system. It is an enhanced replacement of MySQL with more stored engines.

All the above databases have been deployed in an all-in-one configuration mostly using default configurations with some small variations as follows. MongoDB has been configured with sharding and replica support. Cassandra has been deployed over a cluster of two nodes with sharding support. Hypertable has been deployed over Apache Hadoop and built on a cluster providing two nodes for the database and two nodes for the file system. All databases have then been tested in different configurations varying platform-dependent parameters as detailed in Section 5.1.2. Selected databases have been tested and their performance (i.e., throughput) measured using *Yahoo! Cloud Serving Benchmark* (YCSB) framework [16]. YCSB allows to compare the performance of different databases according to several predefined workloads, query types, and query distributions.

5.1.2 Profile and deployment environments

We considered a profile that focuses on property *performance* and assumes a *mostly-read* workload (95% read, 5% update) modeling a typical distributed model. This scenario is aimed at testing databases on their capability of managing distributed computations where data can be collected from different sources. Our workload populates databases with 100k records and submits 10k queries according to the mostly-read profile, using 50 threads simulating parallel requests. The selected profile is such that it can be managed by both SQL and NoSQL databases. This choice permits to test our benchmark in the most difficult scenario in which it needs to rank heterogeneous databases.

We then defined different deployment environments to generate the training and test sets discussed in Section 5.3 and Section 5.4, and to model the impact of the deployment environments and platform-dependent parameters on the weight calculation. We used five different configurations (VM instances) of the deployment environment varying CPU, memory, and storage as follows.

- *Tiny Instance.* It is equipped with 1 vCPU, 20GB of disk, 2x 1GB vNIC, and 1GB of RAM.
- *Small Instance.* It is equipped with 2 vCPU, 50GB of disk, 2x 1GB vNIC, and 2GB of RAM.
- *Medium Instance.* It is equipped with 2 vCPU, 100GB of disk, 2x 1GB vNIC, and 4GB of RAM.
- *Large Instance.* It is equipped with 4 vCPU, 200GB of disk, 2x 1GB vNIC, and 4GB of RAM.
- *Huge Instance.* It is equipped with 8 vCPU, 500GB of disk, 2x 1GB vNIC, and 8GB of RAM.

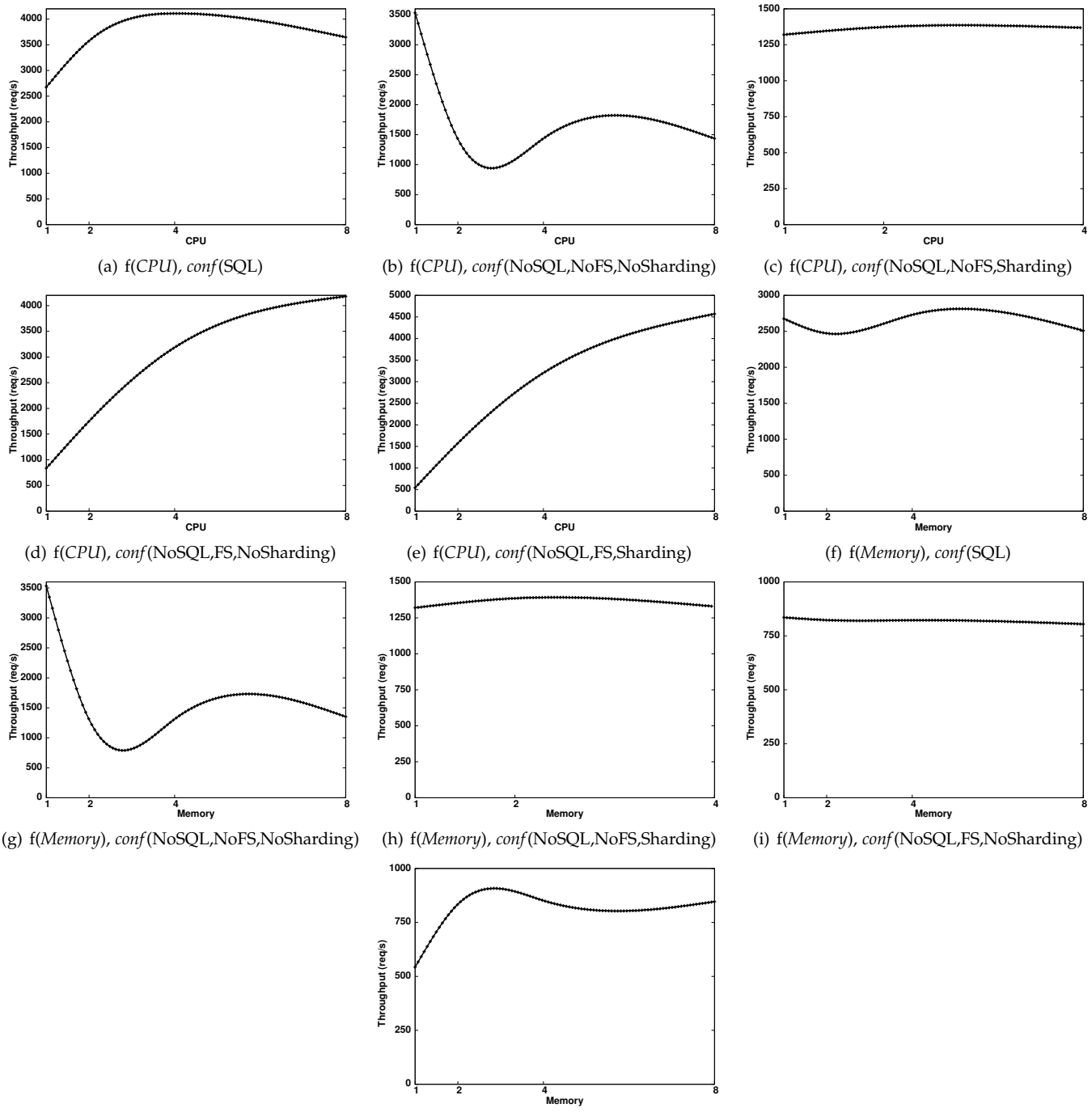
We note that each VM instance installs a Gentoo Linux Stage3 operating system. We deployed all our databases with minimal configuration and optimization efforts as described above in this section.

5.2 Parameter levels

We selected a subset of the parameters in our taxonomy in Figure 1, relevant to our performance evaluation. The selection process was driven by the profile, according to an expert evaluation that identified those parameters which are likely to impact the performance of a database. In this section, we focus on platform-dependent parameters; levels and values for platform-independent parameters, in fact, have been already defined in Section 4.1 (Table 1).

We experimentally evaluated parameters $f(x)$ with $conf_i(Q_i,DB_i,SN_i)$ belonging to attributes *CPU* and *Memory*. Figure 2 describes the functions $f(x)$ modeling levels for each platform-dependent parameter of interest. It is important to note that the more are the number of evaluated parameters "$f(x)$ with $conf_i(Q_i,DB_i,SN_i)$", the higher is the precision of our benchmark. Also, the more detailed are database configurations *conf* (i.e., the broader is the set of parameters in Q_i, DB_i, and SN_i), the better is the precision of $f(x)$. In the extreme case in which complete configurations are available (i.e., specifying all relevant parameters in Q_i, DB_i, and SN_i), the corresponding function $f(x)$ has maximum precision.

In our experiments, we used few high-level configurations to test the strength of our approach in a scenario where values associated with dependent parameters have a level of imprecision. In particular, we distinguished configurations *conf* according to DB type (i.e., either SQL, NoSQL without a distributed file system, NoSQL deployed over a distributed file system) and sharding support (i.e., sharding, no-sharding). Default configurations were used for other parameters. Following the above configurations,



(a) f(*CPU*), *conf*(SQL)
(b) f(*CPU*), *conf*(NoSQL,NoFS,NoSharding)
(c) f(*CPU*), *conf*(NoSQL,NoFS,Sharding)
(d) f(*CPU*), *conf*(NoSQL,FS,NoSharding)
(e) f(*CPU*), *conf*(NoSQL,FS,Sharding)
(f) f(*Memory*), *conf*(SQL)
(g) f(*Memory*), *conf*(NoSQL,NoFS,NoSharding)
(h) f(*Memory*), *conf*(NoSQL,NoFS,Sharding)
(i) f(*Memory*), *conf*(NoSQL,FS,NoSharding)
(j) f(*Memory*), *conf*(NoSQL,FS,Sharding)

Fig. 2. Platform-dependent parameter values

in our experiments we considered MariaDB as the representative of SQL databases, MongoDB as the representative of NoSQL databases without distributed file system, and Hypertable as the representative for NoSQL databases deployed over a distributed file system. We then tested the selected databases deployed in a tiny instance, varying, independently, the main memory in 1GB, 2GB, 4GB, and 8GB, and the CPU in 1vCPU, 2vCPU, 4vCPU, and 8vCPU. We note that some configurations have not been tested due to physical hardware limitations.

Figure 2 shows our results in terms of throughput. As an example, suppose we need to evaluate p="$f(M$=4GB) with *conf*(NoSQL,NoFS,NoSharding)", meaning that a user wants to evaluate a NoSQL database without distributed file system in a sharded configuration, installed in a deployment environment with 4GB of main memory available. According to Figure 2(g), p=0.37, normalized on the maximum throughput. We note that experimental results in this section can be used across different physical environments that have characteristics similar to the environment in this paper (see Section 5.1).

TABLE 2
Training (a) and test (b) sets with reference to parameters in Table 1. Throughput t of databases in the training set (c)

p	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9	d_{10}	d_{11}	d_{12}	d_{13}	d_{14}	d_{15}	d_{16}	d_{17}	d_{18}
p_1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	1	1	1	1	1	1
p_2	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	1	1	1	1	1	1
p_3	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.67	0.67	0.67	0.67	0.67	0.67
p_4	0.1	0.1	0.1	0.1	0.1	0.1	1	1	1	1	1	1	1	1	1	1	1	1
p_5	1	1	1	1	1	1	0.1	0.1	0.1	0.1	0.1	0.1	1	1	1	1	1	1
p_6	0.1	0.1	0.1	0.1	0.1	0.1	1	1	1	1	1	1	1	1	1	1	1	1
p_7	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
p_8	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
p_9	0.1	0.1	0.1	0.1	0.1	0.1	1	1	1	1	1	1	1	1	1	1	1	1
p_{10}	0.1	0.1	0.1	0.1	0.1	0.1	1	1	1	1	1	1	1	1	1	1	1	1
p_{11}	0.67	0.67	0.67	0.67	0.67	0.67	0.1	0.1	0.1	0.1	0.1	0.1	0.67	0.67	0.67	0.67	0.67	0.67
p_{12}	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	1	1	1	1	1	1
p_{13}	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	1	1	1	1	1	1
p_{14}	1	1	1	1	1	1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
p_{15}	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
p_{16}	1	1	1	1	1	1	0.1	0.1	0.1	0.1	0.1	0.1	1	1	1	1	1	1
p_{17}	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	1	1	1	1	1	1
p_{18}	0.1	0.1	0.1	0.1	0.1	0.1	1	1	1	1	1	1	0.1	0.1	0.1	0.1	0.1	0.1
p_{19}	0.1	0.1	0.1	0.1	0.1	0.1	1	1	1	1	1	1	1	1	1	1	1	1
p_{20}	0.1	0.1	0.1	0.1	0.1	0.1	1	1	1	1	1	1	1	1	1	1	1	1
p_{21}	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
p_{22}	1	1	1	1	1	1	0.1	0.1	0.1	1	1	1	0.1	0.1	0.1	0.1	0.1	0.1
p_{23}	1	0.1	0.1	1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.67	0.67	0.67
p_{24}	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
p_{25}	0.1	1	0.1	0.1	1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	1	1	1	1	1	1
p_{26}	0.86	0.33	0.35	0.35	0.33	0.35	0.65	0.87	1	0.65	0.87	1	0.32	0.33	0.33	0.13	0.38	0.78
p_{27}	1	0.39	0.37	0.37	0.38	0.38	0.76	0.7	0.77	0.76	0.7	0.77	1	0.37	0.37	0.15	0.24	0.24

(a)

p	d_1^*	d_2^*	d_3^*	d_4^*	d_5^*	d_6^*	d_7^*	d_8^*	d_9^*	d_{10}^*
p_1	0.1	0.1	0.1	0.1	1	1	0.1	0.1	0.1	0.1
p_2	0.67	0.67	0.1	0.1	1	1	0.1	0.1	0.1	0.1
p_3	0.1	0.1	0.1	0.1	0.67	0.67	0.1	0.1	0.1	0.1
p_4	0.1	0.1	1	1	1	1	0.1	0.1	0.1	0.1
p_5	1	1	0.1	0.1	1	1	0.1	0.1	0.1	0.1
p_6	0.1	0.1	1	1	1	1	0.67	0.67	0.1	0.1
p_7	1	1	1	1	1	1	1	1	0.1	0.1
p_8	1	1	1	1	1	1	1	1	1	1
p_9	0.1	0.1	1	1	1	1	1	1	0.1	0.1
p_{10}	0.1	0.1	1	1	1	1	1	1	0.1	0.1
p_{11}	0.67	0.67	0.1	0.1	0.67	0.67	0.1	0.1	0.1	0.67
p_{12}	0.1	0.1	0.1	0.1	1	1	0.1	0.1	0.1	0.1
p_{13}	0.1	0.1	0.1	0.1	1	1	0.1	0.1	1	1
p_{14}	1	1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
p_{15}	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
p_{16}	1	1	0.1	0.1	1	1	0.1	0.1	1	1
p_{17}	0.1	0.1	0.1	0.1	1	1	0.1	0.1	0.1	0.1
p_{18}	0.1	0.1	1	1	0.1	0.1	1	1	0.1	0.1
p_{19}	0.1	0.1	1	1	1	1	0.1	0.1	1	1
p_{20}	0.1	0.1	1	1	1	1	1	1	0.1	0.1
p_{21}	1	1	1	1	1	1	1	1	1	1
p_{22}	1	1	0.1	1	0.1	0.1	0.1	1	0.1	0.1
p_{23}	0.1	0.1	0.1	0.1	0.1	0.67	0.1	0.1	0.1	0.1
p_{24}	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
p_{25}	0.1	0.67	0.1	0.1	1	1	0.1	0.1	0.1	1
p_{26}	0.31	0.30	0.78	0.78	0.30	0.34	0.30	0.30	0.38	0.34
p_{27}	0.37	0.38	0.77	0.77	0.38	0.24	0.77	0.77	0.23	0.24

(b)

	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9	d_{10}	d_{11}	d_{12}	d_{13}	d_{14}	d_{15}	d_{16}	d_{17}	d_{18}
t	2290.95	1349.35	1465.42	971.91	1348.44	1575.30	2675.94	4775.55	7407.40	2682.40	4980.08	6250.00	835.49	1626.28	3123.05	542.98	1792.11	3264.77

(c)

5.3 Weight selection

We used the methodology described in Section 4.2 to produce our set of weights according to our profile.

To this aim, we first produced our training set by taking three (i.e., MongoDB, MariaDB, Hypertable) out of the five target databases in our experimental set, and varying *i)* the CPU, memory, and storage, according to our deployment environments, and *ii)* database configurations, according to platform-independent parameters in Table 1. Specifically, we considered a total of 18 database configurations: *i) MongoDB* configured varying parameters *Clustering* and *Sharding* and installed on *tiny*, *small*, *medium*, *large*, or *huge* instances (configurations d_1-d_6), *ii) MariaDB* configured varying *Memory Caches*, installed on *tiny*, *small*, or *large* instances (configurations d_7-d_{12}), and *iii) Hypertable* configured varying parameter *Clustering*, and installed on *small*, *medium*, or *huge* instances (configurations d_{13}-d_{18}). The training set is shown in Table 2(a). We note that parameters p_{26} and p_{27} refer to the relevant dependent parameters in attribute *CPU* and *Memory*, respectively. We also note that the more the database configurations are different among them, the more precise are the weights and, hence, our benchmark results. As a consequence, to test the strength of the benchmark, our configurations varied by changing the value of few parameters only. As an example, parameter *Hash* (p_{21}) has value *Supported* in all configurations.

We then experimentally calculated throughput t of databases in the training set (Table 2(c)), following the selected profile. Throughput t_h, with h=1,. . .,18, retrieved for each of the 18 database configurations, has been used to execute our linear regression analysis in Section 4.2 and, in turn, to compute the weights. In particular, for each pair (d_j,d_k) of databases, with $d_j \neq d_k$, a dependent variable $y_i = \frac{t_j}{t_k}$, with $t_j \geq t_k$, is calculated (Equation 1) to generate our linear model (Equation 2). Table 3 shows the weights produced following our methodology (Equation 4). Some weights (e.g., the ones associated with Memtable, BTree, or Hash) are zero, indicating that such parameters are not critical in the computation of the linear regression function, and do not contribute to the final score computation. It is important to note that an improvement of the training set could contribute to an improvement of the cardinality of meaningful weights and, hence, an improvement in the score function.

TABLE 3
Weights

	p	w
p_1	Thrift	0
p_2	Map&Reduce	0
p_3	REST	0
p_4	Cursor	-0.00049
p_5	Collection	0.08026
p_6	Companion SQL DB	0
p_7	Distributed B+ tree	0
p_8	Perfix Hash Table	0
p_9	OR-Junctions	0
p_{10}	AND-Junctions	0
p_{11}	Hot Deploy	0
p_{12}	Data Center Support	0.01046
p_{13}	Column Family	0
p_{14}	Document	0

	p	w
p_{15}	Graph	0.16202
p_{16}	Collection	0
p_{17}	Key Value	0
p_{18}	Relational	0
p_{19}	Memtable	0
p_{20}	Btree	0
p_{21}	Hash	0
p_{22}	Memory Cache	0.00042
p_{23}	Clustering	0.0233
p_{24}	Separate R/W	1
p_{25}	Sharding	-0.03855
p_{26}	CPU	1.39046
p_{27}	Memory	-0.32993

5.4 Putting our methodology in practice

We evaluated our benchmark according to the profile defined in Section 5.1.2. In our evaluation, we considered a single deployment environment (*medium instance* in Section 5.1).

We first generated our test set by taking each of the 5 target databases in different configurations: *i) MongoDB* without support for *sharding* (d_1^*), *ii)* MongoDB with support for *sharding* (d_2^*), *iii) MariaDB* with default *Memory Caches* configuration (d_3^*), *iv) MariaDB* with extended *Memory Caches* configuration (d_4^*), *v) Hypertable* without support for *Clustering* (d_5^*), *vi) Hypertable* with support for *Clustering* (d_6^*), *vii) PostgreSQL* with default *Memory Caches* configuration (d_7^*), *viii) PostgreSQL* with extended *Memory Caches* configuration (d_8^*), *viii) Cassandra* without support for *sharding* (d_9^*), and *ix) Cassandra* with support for *sharding* and *hot deploy* (d_{10}^*).

TABLE 4
Experimental results

Database	Experimental Evaluation		Benchmark Evaluation	
	Throughput t	σ_t	Score $\mathcal{S}(\mathcal{W}_\mathcal{P})\times100$	σ_b
d_4	5042.86	1	85.72	1
d_3	4759.64	2	85.68	2
d_6	1770.85	3	48.35	3
d_5	1686.06	4	36.40	8
d_1	1381.98	5	40.74	5
d_2	1348.44	6	36.82	7
d_7	1339.76	7	18.67	10
d_8	1208.61	8	18.70	9
d_{10}	465.48	9	38.91	6
d_9	262.36	10	47.99	4

Table 2(b) summarizes our test set, describing configurations of each database under evaluation and specifying a value for each of the selected parameters. Again, we note that parameters p_{26} and p_{27} refer to the relevant dependent parameters in attributes *CPU* and *Memory*, respectively.

We then compared our benchmark and methodology in two steps as follows.

1) We experimentally measured throughput t of each database in the test set following the selected profile and deployment environment. Based on t, we calculated a real ranking σ_t of the 10 database configurations.
2) We evaluated each database in the test set according to Definition 4.2 and calculated a scored-based ranking σ_b of the 10 database configurations. In other words, we calculated a synthetic ranking σ_b by applying our methodology using weights in Table 3 and Equation 6 in Definition 4.2.

Table 4 shows the two rankings σ_t and σ_b. To evaluate the quality of our benchmark, we compared them according to two well-know metrics which evaluate the distance between two ordered lists: *i)* the *Spearman's footrule distance* [50] and *ii)* the *Kendall's tau* [51].

The Spearman's footrule distance measures the total displacement between σ_t and σ_b [52]. Let $\sigma_t(i)$ be the rank of database d_i in ranking σ_t and $\sigma_b(i)$ be the rank of database d_i in ranking σ_b. We call σ_t as the identity permutation 1 and define the Spearman's footrule distance as:

$$F(\sigma_b, 1) = F(\sigma_b) = \sum_i |i - \sigma_b(i)|. \quad (7)$$

The Spearman's footrule distance can be normalized in [0,1] by dividing $F(\sigma_b)$ by its maximum value $\frac{n^2}{2}$.

The Kendall's tau distance counts the number of pairwise disagreements between σ_t and σ_b as [52]:

$$K(\sigma_b, 1) = K(\sigma_b) = \sum_{(i,j):i>j} cnt(\sigma_b(i) < \sigma_b(j)) \quad (8)$$

where function *cnt* returns 1 if $\sigma_b(i) < \sigma_b(j)$, 0 otherwise. The Kendall's tau distance can be normalized in [0,1] by dividing $K(\sigma_b)$ by its maximum value $\frac{n(n-1)}{2}$. We note that $F(\sigma_b)$=0 and $K(\sigma_b)$=0 *iff* σ_t=σ_b.

Following our results shown in Table 4, the total displacement between σ_t and σ_b is $F(\sigma_b)$=18, which corresponds to 0.36 when normalized between [0,1]. The number of pairwise disagreements between σ_t and σ_b is instead $K(\sigma_b)$=12, which corresponds to 0.27 when normalized between [0,1]. Both metrics of distance shows that our benchmark preserves a good level of quality for σ_b.

Being configuration independent, our score-based benchmark can be used in conjunction with existing benchmarks to narrow their search space. In this case, the user is not interested in an absolute ranking, but rather in identifying the subset of n databases to be tested. Accordingly, displacement and pairwise disagreement of elements near the head of σ_t are more important. To this aim, we rely on *weighted* Spearman's footrule (F_δ) and Kendall's tau (K_δ) as discussed in [52]. Let s_i>0 be the weight of an element i∈σ_t. We define the weighted Spearman's footrule as

$$F_s(\sigma_b) = \sum_i s_i |i - \sigma_b(i)|, \quad (9)$$

where each displacement is scaled by the weight of the displaced element.

We then define the Kendall's tau (K_δ) as

$$K_s(\sigma_b) = \sum_{i>j} s_i s_j cnt(\sigma_b(i) < \sigma_b(j)), \quad (10)$$

where an inversion of elements i and j presents a penalty proportional to their product, that is, $s_i s_j$. The weighted Spearman's footrule distance (the Kendall's tau distance, resp.) can be normalized in [0,1] by dividing $F_s(\sigma_b)$ ($K_\delta(\sigma_b)$, resp.) by its maximum value.

To model the importance of the positions near the head of σ_t, we set the weights s as s_1=10, s_2=9, s_3=8, s_4=7, s_5=6, s_6=5, s_7=4, s_8=3, s_9=2, s_{10}=1. Our results are shown in Table 4. The total displacement between σ_t and σ_b is $F_s(\sigma_b)$=78, which corresponds to 0.26 when normalized between [0,1]. The number of pairwise disagreements between σ_t and σ_b is instead $K_s(\sigma_b)$=211, which corresponds to 0.16 when normalized between [0,1]. We note that both weighted metrics $F_s(\sigma_b)$ and $K_s(\sigma_b)$ present lower values (i.e., higher quality for σ_b) with respect to the corresponding unweighted version $F(\sigma_b)$ and $K(\sigma_b)$. We also note that in case the weights are distributed to give even more relevance to the first half of elements in σ_t, the quality of σ_b increases or, in other words, the values $F(\sigma_b)$ and $K(\sigma_b)$ decrease.

5.5 Analysis of the results

Our experiments show that our benchmark provides a good level of quality also when the training set and the experimental evaluation of dependent parameters are designed to introduce a not-negligible level of imprecision. In fact, we defined a training set with a limited variety in terms of configuration parameters, and evaluated the impact of dependent parameters using abstract database configurations. Clearly, the more the precision of the training set and of experiments on dependent parameters, the higher the quality and precision of our benchmark.

According to our results, the normalized Kendall's tau distance between σ_t and σ_b is 0.36, while the normalized Spearman's footrule is 0.26. Also, the results show a better quality for σ_b when we penalize more total displacement and pairwise disagreements involving elements near the head of σ_t. By assigning linear weights s_1=10, s_2=9, s_3=8,

s_4=7, s_5=6, s_6=5, s_7=4, s_8=3, s_9=2, s_{10}=1, the normalized Kendall's tau becomes 0.26, with an increase in quality of 27.8%, while the normalized Spearman's footrule becomes 0.16, with an increase in quality of 38.5%. Moreover, 4 out of the first 5 databases in σ_t are kept in the same position in σ_b, corresponding to a precision of 80%. We observe that although d_5, the only database not maintained in the top-5 ranking, is moved from the 4th position to 8th position, its score (36.40) is very similar to the ones of databases in the 6th (38.91) and 7th (36.82) positions. The latter result shows that different databases with similar configurations can be ranked with similar scores, which make the decision about which database to select hard and often random. This is particularly relevant when similar scores are provided for top-ranked databases. In this case, the adoption of our benchmark in conjunction with existing benchmarks could represent a proper balance between overhead and quality.

We remark that additional experiments have been done widening the set of evaluated databases, changing the deployment environment, and changing the workload. Results are available at http://tinyurl.com/onvfg4o for interested readers.

6 Discussion

We discuss pros and cons of our score-based benchmark with respect to several different aspects, including the four main requirements (i.e., relevance, scalability, portability, simpleness) specified by Jim Gray in [20].

Simplicity. According to [20] "*A domain-specific benchmark must be understandable, otherwise it will lack credibility*". Our benchmark requires final users to only define their profile (i.e., target property and expected workload) and database configurations to retrieve a score-based ranking with no need of real testing/deployment. Database configurations include high-level characteristics of the databases target of the evaluation and low-level details of their deployment environment. This information should be at disposal of any users starting a comparative database evaluation, and used to *i)* select the set of weights generated in Section 4.2 and *ii)* calculate Equation 6 in Definition 4.2. In the line with community-based assessment methods [53], a library of weights can be defined for different combinations of user's profiles, and managed by a community of experts. As discussed in Section 4, profiles can also specify details on the physical infrastructure used for weight calculation. This would increase the quality of our methodology. We note that our benchmark might require, in case no suitable weights are available for the selected domain, to manage the approach in Section 4.2 for weight calculation. Tools can be provided to simplify weight calculation; as an example, an excel file implementing our methodology is available for download at http://tinyurl.com/pzf6hh7. In case users decide to build their own weights, as for existing benchmarks, they also need to deploy (a subset of) target databases in their final infrastructure and experimentally evaluate them.

Relevance. According to [20] "*A domain-specific benchmark must measure the peak performance and price/performance of systems when performing typical operations within that problem domain*". Having a profile $\Pi_{pr,wl}$ specified by the user, our benchmark supports the relevance requirement. Users in fact can specify the relevant workload to verify database properties in the given domain. Also, our approach is not restricted to traditional performance properties and can be extended to certify different properties (e.g., availability, consistency) with no additional effort on the final users. Similarly to existing benchmarks, lack of weights for the specified profile either requires users to build their own weights or to consider weights which do not fully fit the considered domain.

Portability. According to [20] "*It should be easy to implement the benchmark on many different systems and architectures*". Our benchmark is configuration independent and therefore supports the portability requirement. The availability of weights calculated by database experts for a given domain allows final users to evaluate target databases independently from database and environment configurations. Configurations are specified as parameter levels by users without requiring any deployment/implementation activity. Similarly to existing benchmarks, lack of weights for the considered domain either requires users to build their own weights or to consider weights which do not fully fit the considered domain.

Scalability. According to [20] "*A domain-specific benchmark should apply to small and large computer systems. It should be possible to scale the benchmark up to larger systems, and to parallel computer systems as computer performance and architecture evolve*". Being portable and configuration independent, our benchmark supports the scalability requirement. Scalability is in fact managed at the level of expert evaluation for weight calculation. Similarly to existing benchmarks, lack of weights for the considered domain either requires users to build their own weights or to consider weights which do not fully fit the considered domain.

Flexibility/Extensibility. Our benchmark is fully customizable on the users' needs, with no impact on the proposed methodology for weights and score calculation. First, the taxonomy in Section 3, as well as the levels in Section 4.1, can be extended/modified to address continuous evolution of distributed databases. Also, weights can be incrementally modified on the basis of new experiments and/or improved training sets. Finally, the approach for weight calculation based on linear regression analysis can be enhanced to more complex approaches, such as for instance machine learning techniques with l1regularization. In summary, the quality of the benchmark can be increased by adding more variety on the considered independent parameters, more instances in the training set, more experiments on the dependent parameters, and by selecting the proper technique for weight calculation.

Interoperability. Our benchmark can be used as a standalone approach generating an absolute ranking of databases, as well as in conjunction with existing benchmarks to narrow their search space. In the latter case, the best n databases in the ranking are evaluated by means of traditional benchmarks.

Heterogeneity. Our benchmark supports comparison of both SQL and NoSQL databases.

In summary, to the best of our knowledge, differently from the solution in this paper, existing benchmarks *i)* do not support comparison of SQL and NoSQL databases (only [25] compares the popularity of SQL and NoSQL database in a single ranking), *ii)* require high competences and put high overhead on final users, *iii)* require experimental evaluation of each candidate database, *iv)* consider some specific scenarios in terms of workload, deployment environment, making benchmark results invalid in case of changes in the real user scenario, *v)* are architecture dependent.

7 Conclusions

We presented a taxonomy and a method for evaluating and comparing distributed databases and their performance. Our approach aims at providing a generic and configuration-independent score-based benchmark for NoSQL databases, which can also be applied to SQL databases. It departs from the assumptions that each benchmark should apply to a specific scenario and configuration, and should be physically executed in the user's infrastructure. Rather, all users have to do is defining a profile that specifies the property to be evaluated and a workload of interest, and selecting a set of pre-computed weights to compare a set of databases with known configurations. Our approach can work both as a standalone benchmark and as a supporting tool to reduce the search space of other benchmarks, and in turn their overheads. We claim that our method can pave the way to a future community-oriented effort that will manage the taxonomy lifecycle, and the definition of new profiles and corresponding weights for the comparison of different databases in different scenarios.

Acknowledgments

This work was partly supported by Telecom Italia S.p.A. (contract n. 9/2013), the Italian MIUR project SecurityHorizons (c.n. 2010XSEMLC) and by the EU-funded project CUMULUS (contract n. FP7-318580).

References

[1] European Commission, *Helping SMEs Fish the Big Data Ocean*, July 2014, http://ec.europa.eu/digital-agenda/en/news/helping-smes-fish-big-data-ocean.

[2] M. J. Carey, D. J. DeWitt, and J. F. Naughton, "The 007 benchmark," in *Proc. of the 1993 ACM SIGMOD International Conference on Management of Data (SIGMOD 1993)*, Washington, DC, USA, May 1993.

[3] J. Gray, *Benchmark Handbook: For Database and Transaction Processing Systems*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1992.

[4] A. Schmidt, F. Waas, M. Kersten, M. J. Carey, I. Manolescu, and R. Busse, "Xmark: A benchmark for xml data management," in *Proc. of the 28th International Conference on Very Large Data Bases (VLDB 2002)*, Hong Kong, China, August 2002.

[5] *Linux Benchmark Suite Homepage*, http://lbs.sourceforge.net/, Accessed September 2015.

[6] Standard Performance Evaluation Corporation, *SPEC's Benchmarks*, https://www.spec.org/benchmarks.html, Accessed September 2015.

[7] Transaction Processing Performance Council (TPC), *TPC Benchmarks*, http://www.tpc.org/information/benchmarks.asp, Accessed September 2015.

[8] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, and B. Falsafi, "Clearing the clouds: A study of emerging scale-out workloads on modern hardware," in *Proc. of the 17th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2012)*, London, UK, March 2012.

[9] S. Islam, K. Lee, A. Fekete, and A. Liu, "How a consumer can measure elasticity for cloud platforms," in *Proc. of the 3rd ACM/SPEC International Conference on Performance Engineering (ICPE 2012)*, Boston, MA, USA, April 2012.

[10] T. Rabl, M. Frank, H. M. Sergieh, and H. Kosch, "A data generator for cloud-scale benchmarking," in *Proc. of the 2nd TPC Technology Conference on Performance Evaluation, Measurement and Characterization of Complex Systems (TPCTC 2010)*, Singapore, September 2010.

[11] A. Turner, A. Fox, J. Payne, and H. S. Kim, "C-mart: Benchmarking the cloud," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 6, pp. 1256–1266, 2013.

[12] R. Cattell, "Scalable sql and nosql data stores," *ACM SIGMOD Record*, vol. 39, no. 4, pp. 12–27, December 2010.

[13] R. Hecht and S. Jablonski, "Nosql evaluation: A use case oriented survey," in *Proc. of the 2011 International Conference on Cloud and Service Computing (CSC 2011)*, Hong Kong, China, December 2011.

[14] S. Sakr, A. Liu, D. Batista, and M. Alomari, "A survey of large scale data management approaches in cloud environments," *IEEE Communications Surveys Tutorials*, vol. 13, no. 3, pp. 311–336, 2011.

[15] T. G. Armstrong, V. Ponnekanti, D. Borthakur, and M. Callaghan, "Linkbench: A database benchmark based on the facebook social graph," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data (SIGMOD 2013)*, New York, NY, USA, June 2013.

[16] B. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with ycsb," in *Proc. of the ACM Symposium on Cloud Computing (SoCC 2010)*, Indianapolis, IN, USA, March 2010.

[17] T. Dory, B. Mejías, P. Van Roy, and N.-L. Tran, "Comparative elasticity and scalability measurements of cloud databases," 2011, http://www.nosqlbenchmarking.com/wp-content/uploads/2011/05/paper.pdf.

[18] E. Dede, M. Govindaraju, D. Gunter, R. S. Canon, and L. Ramakrishnan, "Performance evaluation of a mongodb and hadoop platform for scientific data analysis," in *Proc. of the 4th ACM Workshop on Scientific Cloud Computing (Science Cloud 2013)*, New York, NY, USA, June 2013.

[19] A. Ghazal, T. Rabl, M. Hu, F. Raab, M. Poess, A. Crolotte, and H.-A. Jacobsen, "Bigbench: Towards an industry standard benchmark for big data analytics," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data (SIGMOD 2013)*, New York, NY, USA, June 2013.

[20] J. Gray, "Database and transaction processing performance handbook," in *Benchmark Handbook: For Database and Transaction Processing Systems*, J. Gray, Ed. Morgan Kaufmann Publishers Inc., 1992.

[21] Y. Chen, A. Ganapathi, R. Griffith, and R. Katz, "The case for evaluating mapreduce performance using workload suites," in *Proc. of the 19th IEEE International Symposium on Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS 2011)*, Singapore, July 2011.

[22] D. Laney, *3D Data Management: Controlling Data Volume, Velocity and Variety*, Meta Group Inc., http://blogs.gartner.com/doug-laney/files/2012/01/ad949-3D-Data-Management-Controlling-Data-Volume-Velocity-and-Variety.pdf.

[23] R. Lomotey and R. Deters, "Analytics-as-a-service framework for terms association mining in unstructured data," *International Journal of Business Process Integration and Management (IJBPIM)*, vol. 7, no. 1, pp. 49–61, 2014.

[24] Y. Li and S. Manoharan, "A performance comparison of sql and nosql databases," in *Proc. of the IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM 2013)*, Victoria, Canada, August 2013.

[25] *DB-Engines Ranking*, http://db-engines.com/en/ranking.

[26] Z. Li, L. O'Brien, H. Zhang, and R. Cai, "On a catalogue of metrics for evaluating commercial cloud services," in *Proc. of the ACM/IEEE 13th International Conference on Grid Computing (GRID 2012)*, Beijing, China, September 2012.

[27] *Liquid Benchmarking Platform*, http://wiki.liquidbenchmark.net/doku.php, Accessed September 2015.

[28] CloudSpectator, *Cloud Computing Performance a Comparative Analysis of 5 Large Cloud IaaS Providers*, June 2013, http://tinyurl.com/pcfe8n8, Accessed September 2015.

[29] L. M. Vaquero, L. Rodero-Merino, and R. Buyya, "Dynamically scaling applications in the cloud," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 1, pp. 45–52, January 2011.

[30] A. Iosup, S. Ostermann, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema, "Performance analysis of cloud computing services for many-tasks scientific computing," *IEEE Transactions on Parallel and Distributed Systems (TDPS)*, vol. 22, pp. 931–945, June 2011.

[31] K. Salah and R. Boutaba, "Estimating service response time for elastic cloud applications," in *Proc. of the 1st IEEE International Conference on Cloud Networking (CLOUDNET 2012)*, Paris, France, November 2012.

[32] V. Stantchev, "Performance evaluation of cloud computing offerings," in *Proc. of the 3rd International Conference on Advanced Engineering Computing and Applications in Sciences (ADVCOMP 2009)*, Sliema, Malta, October 2009.

[33] J. Espadas, A. Molina, G. Jiménez, M. Molina, R. Ramírez, and D. Concha, "A tenant-based resource allocation model for scaling software-as-a-service applications over cloud computing infrastructures," *Future Generation Computer Systems*, vol. 29, no. 1, pp. 273–286, 2013.

[34] G. Copil, D. Moldovan, H.-L.Truong, and S. Dustdar, "Multi-level elasticity control of cloud services," in *Service-Oriented Computing*, ser. Lecture Notes in Computer Science, S. Basu, C. Pautasso, L. Zhang, and X. Fu, Eds. Springer Berlin Heidelberg, 2013, vol. 8274, pp. 429–436.

[35] D. Tsoumakos, I. Konstantinou, C. Boumpouka, S. Sioutas, and N. Koziris, "Automated, elastic resource provisioning for nosql clusters using tiramola," in *Proc. of the 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2013)*, Delft, The Netherlands, May 2013.

[36] I. Konstantinou, E. Angelou, C. Boumpouka, D. Tsoumakos, and N. Koziris, "On the elasticity of nosql databases over cloud management platforms," in *Proc. of the 20th ACM Conference on Information and Knowledge Management (CIKM 2011)*, Glasgow, Scotland, October 2011.

[37] F. Cruz, F. Maia, M. Matos, R. Oliveira, J. a. Paulo, J. Pereira, and R. Vilaça, "Met: Workload aware elasticity for nosql," in *Proc. of EuroSys 2013*, Prague, Czech Republic, April 2013.

[38] Y. Zhang, Z. Wang, B. Gao, C. Guo, W. Sun, and X. Li, "An effective heuristic for on-line tenant placement problem in SaaS," in *Proc. of 8th IEEE International Conference on Web Services (ICWS 2010)*, Miami, FL, USA, July 2010.

[39] N. Vasić, D. Novaković, S. Miučin, D. Kostić, and R. Bianchini, "Dejavu: Accelerating resource allocation in virtualized environments," in *Proc. of the 17th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2012)*, London, UK, March 2012.

[40] R. Krebs, A. Wert, and S. Kounev, "Multi-tenancy performance benchmark for web application platforms," in *Web Engineering*, ser. Lecture Notes in Computer Science, F. Daniel, P. Dolog, and Q. Li, Eds. Springer Berlin Heidelberg, 2013, vol. 7977, pp. 424 – 438.

[41] Z. Li, L. O'Brien, H. Zhang, and R. Cai, "On a catalogue of metrics for evaluating commercial cloud services," in *Proc. of the ACM/IEEE 13th International Conference on Grid Computing (GRID 2012)*, Beijing, China, September 2012.

[42] A. Ali-Eldin, J. Tordsson, and E. Elmroth, "An adaptive hybrid elasticity controller for cloud infrastructures," in *Proc. of the IEEE Network Operations and Management Symposium (NOMS 2012)*, Maui, HI, USA, April 2012.

[43] J. Fito, I. Goiri, and J. Guitart, "Sla-driven elastic cloud hosting provider," in *Proc. of the 18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP 2010)*, Pisa, Italy, February 2010.

[44] M. Klems, A. Silberstein, J. Chen, M. Mortazavi, S. A. Albert, P. Narayan, A. Tumbde, and B. Cooper, "The yahoo!: Cloud datastore load balancer," in *Proc. of the Fourth International Workshop on Cloud Data Management (CloudDB 2012)*, Maui, HI, USA, October-November 2012.

[45] C. Curino, E. Jones, Y. Zhang, and S. Madden, "Schism: A workload-driven approach to database replication and partitioning," *Proceedings of the VLDB Endowment*, vol. 3, no. 1–2, September 2010.

[46] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, 2004.

[47] T. Hoff, "A yes for a nosql taxonomy," 2009, http://highscalability.com/blog/2009/11/5/a-yes-for-a-nosql-taxonomy.html.

[48] S. Yen, "Nosql is a horseless carriage," 2009, http://dl.getdropbox.com/u/2075876/nosql-steve-yen.pdf.

[49] B. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with ycsb," in *Proc. of the ACM Symposium on Cloud Computing (SoCC 2010)*, Indianapolis, IN, USA, March 2010.

[50] C. Spearman, "The proof and measurement of association between two things, by c. spearman, 1904," *The American journal of psychology*, vol. 100, no. 3-4, pp. 441–471, 1987.

[51] M. Kendall, "A new measure of rank correlation." *Biometrika*, vol. 30, no. 1/2, pp. 81–93, 1938.

[52] R. Kumar and S. Vassilvitskii, "Generalized distances between rankings," in *Proc. of the 19th International Conference on World Wide Web (WWW 2010)*, Raleigh, NC, USA, April 2010.

[53] O. Alonso and R. Baeza-Yates, "Design and implementation of relevance assessments using crowdsourcing," in *Advances in Information Retrieval*, P. Clough, C. Foley, C. Gurrin, G. Jones, W. Kraaij, H. Lee, and V. Mudoch, Eds. Springer Berlin Heidelberg, 2011, pp. 153–164.

Claudio A. Ardagna is an Associate Professor at the Dipartimento di Informatica, Università degli Studi di Milano, Italy. His research interests are in the area of cloud security and performance. He is the recipient of the ERCIM STM WG 2009 Award for the Best PhD Thesis on Security and Trust Management. He has co-authored the Springer book "Open Source Systems Security Certification". The URL for his web page is http://www.di.unimi.it/ardagna

Ernesto Damiani is a Full Professor at the Università degli Studi di Milano and the leader of the Big Data Initiative at the EBTIC/Khalifa University in Abu Dhabi, UAE. He was a recipient of the Chester-Sall Award from the IEEE IES Society (2007). He was named ACM Distinguished Scientist (2008) and he received the IFIP TC2 Outstanding Contributions Award (2012). He is the Vice-Chair of the IEEE Technical Committee on Industrial Informatics.

Fulvio Frati holds an administrative position at the Dipartimento di Informatica, Università degli Studi di Milano. He is author of many international scientific publications in the field of Open Source, Service Oriented Architecture, Collaboration Environment, and Software Development Process Monitoring and Modeling. He has served as PC member and publication chair of many International conferences and workshops.

Davide Rebeccani is currently IT Manager at the Dipartimento di Informatica, Università degli Studi di Milano. His interests are in the areas of operating systems, system and network administration, cloud computing platforms, and security of distributed systems. Since 1997, he has worked on several types of open source operating systems such as Linux, FreeBSD, OpenBSD, and NetBSD. Currently, he is working on cloud computing platforms and NoSQL databases.