TDM: A Tensor Data Model for Logical Data Independence in Polystore Systems

Eric Leclercq[1] and Marinette Savonnet[1]

LE2I EA 7508 - University of Bourgogne - Dijon 20178, France
{eric.leclercq,marinette.savonnet}@u-bourgogne.fr

Abstract. This paper presents a Tensor Data Model to carry out logical data independence in polystore systems. TDM is an expressive model that can link different data models of different data stores and simplifies data transformations by expressing them by means of operators whose semantics are clearly defined. Our contribution is the definition of a data model based on tensors for which we add the notions of typed schema using associative arrays. We describe a set of operators and we show how the model constructs take place in a mediator/wrapper like architecture.

Keywords: polystore · data model · logical data independence · tensor.

1 Introduction and motivations

In a globalized economy, driven by digital technologies, data become an element of added value and wealth. Beyond the volume, data are also more and more diverse in their production mode and use. All sectors of the economy are undergoing a deep transformation of their business. Banks must be able to trace and detect potential risks for their clients, to predict evolution of stock markets. Retail must be able to optimize tour management using historical data deliveries. In the energy sector, the implementation of remote control tools, smart-grids can be used to optimize and adapt networks delivery to users needs. In the environmental sector, data can be used to detect pollution phenomena, anticipate the risks of flooding. In marketing, social networks data are used to discover communities, opinion leaders or to detect and study the propagation of fake news.

All these domains use data grounded in different models such as complex networks, times series, grids, cubed sphere, multi-layer networks, etc. In order to study these data in depth, i.e. to go through data, information and knowledge layers, several analyses are made. Analyses usually require to use data from different sources in an integrated way and are performed by algorithms which have different theoretical foundations such as graph theory, linear algebra, statistical models, Markov models and so on. For example, many algorithms for detecting communities use a graph represented as an adjacency matrix associated with a random walk or a density measure. Recommendation systems built with machine learning techniques frequently use linear algebra such as singular value

decomposition (SVD)[5] and/or alternating least squares (ALS) [53]. To develop predictive models that determine relationships between graph entities, identify behaviors or detect anomalies or events in time series, statistical models such as covariance matrix are needful.

Furthermore, subjects or goals of modern data-intensive applications are not always well-defined. This is especially true when the aim of analyses is scientific research or when the phenomenon being analyzed is poorly understood. These characteristics are not typically observed in Business Intelligence where enterprise data semantics is usually well-defined. For example, there is no ambiguity about a person who is a customer and about its features. However, in a more general context, a single algorithm is not enough to analyze data and to ensure the veracity of the results. For example, social scientists will gather a set of results from different community detection algorithms applied on Twitter data to establish a body of evidence and then conduct a qualitative study.

Anyway, forcing varied data to fit into a single database could lead to performance problems and be a hindrance to setting up analyses. As stated by Stonebraker in [48, 47] "one size fits all" is not a solution for modern data-intensive applications. Likewise Ghosh explains in [19], storing data the way it is used in an application simplifies programming and makes it easier to decentralize data processing. For these reasons, a number of research projects are shifting the focus to build system on multiple data stores using multiple data models.

In-database analysis is another important issue as it can reduce complex and costly data transformations (i.e. features selection, export and convert data) before applying analysis algorithms. Recent algorithms are rarely implemented in DBMS and matrix operations and associated factorizations [38, 23] are not directly supported by traditional storage systems. For graph analysis tools, only a few NoSQL systems like Neo4j support a small set of algorithms[1]. However, Neo4j does not allow to manage very large amount of data with attributes as the column-oriented systems would do [24]. The situation is almost similar for machine learning algorithms and tools. Only some recent systems such as Vertica[2] or SciDB[3] support standard machine learning algorithms as black-box. On the other hand some tool-boxes such as TensorFlow[4], Theano[5], Keras[6] or MLlib in Apache Spark[7] have been developed to design machine learning tools using data structures close to algorithms. As a result these systems require to develop complex, hard to reuse and often error-prone programs for loading and transforming data [22, 1].

We propose to revisit logical data independence in the context of polystore to develop an approach close to in-database analysis that integrates analysis

[1] https://neo4j.com/developer/graph-algorithms/
[2] https://www.vertica.com/product/database-machine-learning/
[3] https://www.paradigm4.com/
[4] https://www.tensorflow.org/
[5] http://deeplearning.net/software/theano/
[6] https://keras.io/
[7] https://spark.apache.org/mllib/

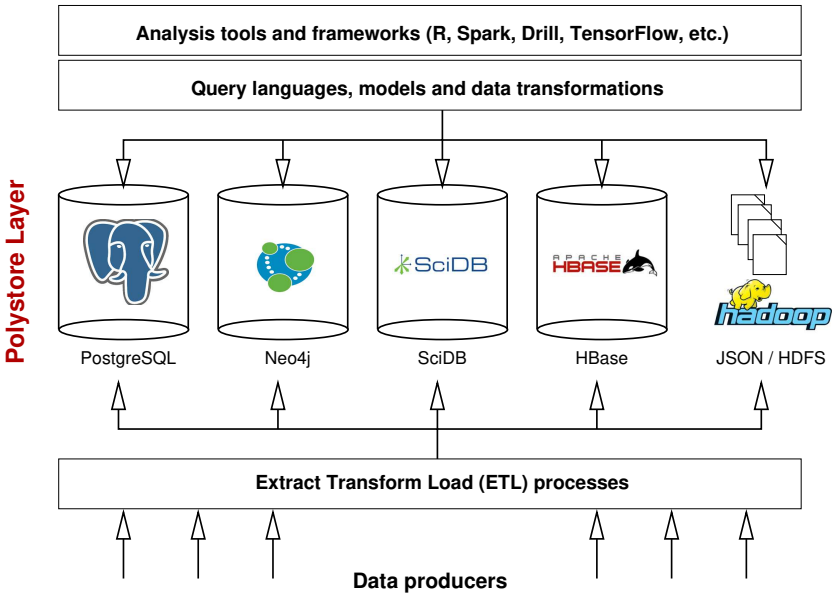

Fig. 1. Outline of the approach

tools such as Spark, R, Drill, TensorFlow in a loosely coupled architecture. Our approach allows users to quickly feed algorithms with data coming from several databases by reducing models and data transformations (figure 1).

The remainder of the paper is organized as follows. While section 2 discusses about multi-paradigm storage systems, section 3 presents the tensor based data model and its operators. Section 4 describes the software architecture and different experiments and results obtained in TEP 2017 project which studies the use of Twitter during the French presidential election in 2017.

2 Related works on multi-paradigm storage systems

The problem of accessing to heterogeneous data sources has been addressed for many years by research communities in schema integration and multi-database system [44]. Big Data oriented storage systems like HDFS and NoSQL systems, which have been mature for several years, have changed the heterogeneous data access issues [17]. As a result, several research projects have been inspired by previous work on distributed databases [44] in order to take advantage of a federation of specialized storage systems with different models[8]. Multi-paradigm data storage relies on multiple data storage technologies, chosen according to the way data is used by applications and/or by algorithms [45]. The problem is

[8] `http://wp.sigmod.org/?p=1629`

magnified by some other facts: 1) NoSQL systems do not always have a well-established separation between logic model and physical model; 2) to achieve flexibility new systems do not necessarily provide a well-defined schema.

2.1 A taxonomy

In [49] authors propose a survey of such systems and a taxonomy in four classes:

- **federated database systems** as collection of homogeneous data stores and a single query interface;
- **polyglot systems** as a collection of homogeneous data stores with multiple query interfaces;
- **multistore systems** as a collection of heterogeneous data stores with a single query interface;
- **polystore systems** as a collection of heterogeneous data stores with multiple query interfaces.

In order to have more significant groups of systems we adopt a slightly different classification that replaces federated database systems by a more specific class of pragmatic systems using a common query language. So, our updated classification is based on models and languages by: 1) considering multi-database query language approach [40] instead of federated systems to better represent the autonomy of data sources and existing enterprise-oriented systems; 2) replacing homogeneity of data model systems by isomorphic models[9], for example for JSON and the relational model [9, 15] and; 3) instead of using query interface or query engine terms as a criterion we prefer query language. According to these criteria our classification is (table 1): multi-database query language (unique language), polyglot system including data models isomorphic to relational model (with multiple languages), multistore, and polystore. For each of these classes we describe some of the most significant representative systems.

language / model	single	multiple
single or isomorphic	multibase	polyglot
multiple	multistore	polystore

Table 1. Classification of multi-paradigm storage approaches

[9] To be isomorphic two data models must allow two way transformations at the structure level but also support equivalence between sets of operators. For example graph data model and relational data model are not isomorphic because relational data model with relational algebra do not support directly transitive closure.

2.2 Representative systems

Spark SQL[10] is the major representative of multidatabase query language. It allows to query structured data from relational-like data sources (JDBC, JSON, Parquet, etc.) in Spark programs, using SQL. Apache Drill[11] is similar to Spark without having a very large support of analysis algorithms as Spark does with MLlib and GraphX.

According to our classification, CloudMdsQL [35] is more a polyglot system than a multistore system as suggested by the title of one of their articles published before the first taxonomy proposal. CloudMdsQL is a functional SQL-like language, designed for querying multiple data store engines (relational or column-store) within a query that may contain sub-queries to each data store's native query interface. SQL++ which is a part of the FORWARD platform[12], is a semi-structured query language that encompasses both SQL and JSON [41, 42].

HadoopDB [2] coupled to Hive[13] is a multistore, it uses the map-reduce paradigm to push data access operations on multiple data stores. D4M (Dynamic Distributed Dimensional Data Model) [29] is a multistore that provides a well founded mathematical interface to tuple stores. D4M allows matrix operations and linear algebra operators composition and applies them to the tuple stores. D4M reduces the autonomy of data stores to achieve a high level of performance [30].

The BigDAWG system [18, 16] is a polystore allowing to write multi-database queries with reference to islands of information, each corresponding to a type of data model (PostgreSQL, SciDB and Accumulo). Myria [52] supports multiple data stores as well as different data computing systems such as Spark. It supports SciDB for array processing, RDBMS, HDFS. The RACO (Relational Algebra COmpiler) acts as a query optimizer and processor for MyriaL language. Myria also supports user data functions in different other languages such as Python. Morpheus [3] is a polystore approach, implemented using Apache Spark, that focuses on Cypher query language instead of SQL and takes advantage of graph analysis algorithms implemented in Neo4j.

2.3 Discussion

Polystores are designed to make the best use of the data models, combining systems by unification with language. Tool-box approaches (TensorFlow, Theano) use data structures close to algorithms but they do not supply storage mechanism. The MLog system [39] is an hybrid approach which defines a tensor data model with operators and study optimization techniques for queries over TensorFlow.

[10] `https://spark.apache.org/sql/`
[11] `https://drill.apache.org/`
[12] `http://forward.ucsd.edu/`
[13] `https://hive.apache.org/`

Several kinds of data analytics platforms have also been defined in the last few years [46]. They are usually an aggregation of existing technologies and can be classified in computation-centric architecture or data-centric architecture. Two main typical architectures are data analytics stacks and data lakes.

New data analytics stacks have emerged as infrastructure for giving access to data stores and enabling data processing workflows. The Berkeley Data Analytics Stack (BDAS) from the AMPLAb project[14] is a multi-layer architecture that provides multiple storage layer (multistore) using Alluxio[15] and data processing using Apache Spark ecosystem.

The IT industry uses the metaphor of data lake to define shared data environment consisting of multiple repositories. A data lake provides data to a variety of processing systems including streaming. The solutions are mature and there are products on the market such as Microsoft Azure Data Lake[16], IBM data lake [17]. Alluxio included in BDAS is also a data lake system.

The ANSI/SPARC architecture [12] characterizes classical data management systems (relational, object-oriented) from a logical point of view by proposing a 3-layer decomposition that reflects the abstraction levels of data: (i) the external data schemata describe the different external views over data dedicated to end-users or applications; (ii) the logical or conceptual schema describes entities and relationships among them, including integrity constraints and (iii) the physical schema describes the storage and the organization of data. As for operating systems or network protocols, Härder and Reuter [20, 21] have proposed a decomposition of the functional architecture of a DBMS in 5 layers: i) file management that operates on blocks and files, ii) propagation controls that define and manage segments and pages, iii) records and access path management that works on access path and physical records, iv) record oriented navigational access that describes records, sets, hierarchies and v) non-procedural or algebraic access that defines tuples, relations, views and operators for logical schema description and data retrieval. But most of RDBMS rather use less layer following System R [6] which defines two layers: 1) the Relational Storage System (RSS) with a Relational Storage Interface (RSI) which handles access to tuples and manages devices, space, allocation, storage buffers, transaction consistency and locking as well as indexes; 2) the Relational Data System (RDS) with a Relational Data Interface (RDI) provides authorization, integrity enforcement, and support views of data as well as a definition, manipulation and query language. The RDS also maintains the catalogs of names to establish correspondences with internal names in RSS.

NoSQL systems, beyond their models differences, exhibit a common characteristic with respect to the architecture: the external and logic levels disappear [50]. As a consequence the applications are close to the physical level with no real logical independence between programs and data. Moreover, due to the nature

[14] `https://amplab.cs.berkeley.edu/software/`
[15] `http://www.alluxio.org/`
[16] `https://azure.microsoft.com/en-us/services/data-lake-analytics/`
[17] `https://www.ibm.com/analytics/data-lake`

of schema-less NoSQL systems, the source code contains implicit assumptions about the data schema. These drawbacks make it difficult to set up a data curation process. The ingestion phase can be done quite easily (i.e. a feature puts forward by data lake vendors) but the data transformation, schema integration, data cleaning and entity consolidation are heavily hindred by the lack of logical and external schemata.

Our approach is a top-down approach that favors the notion of model and applications in contrast to bottom-up approaches that are guided by performance and query optimization. The Tensor Data Model acts as views over data sources, and aims at quickly feed algorithms with data coming from several sources by reducing models and data transformations. Our focus is on the theoretical foundation of the model and the algebraic structures of its operators before studying their implementation. It is obvious that these two points of view must be studied jointly to build an operational system.

3 Core concepts of TDM

This section addresses the definition of a Tensor Data Model (TDM), starting with the tensor mathematical object, we add it the notion of typed schema using associative arrays and we define a set of data manipulation operations. We also study mappings between TDM and others data models.

Tensors are very general abstract mathematical objects which can be considered according to various points of view. A Tensor can be seen as a multi-linear applications, as the result of the tensor product, as an hypermatrix. We will use the definition of a tensor as an element of the set of the functions from the product of N sets $I_j, j = 1, \ldots, N$ to $\mathbb{R} : \boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$, N is the number of dimension of the tensor or its order or its mode. In a more general definition, a tensor is a family of values indexed by N finite sets. A tensor is often treated as a generalized matrix, 0-order tensor is a scalar, 1-order one is a vector, 2-order one is a matrix, tensors of order 3 or higher are called higher-order tensors.

Tensor operations, by analogy with the operations on matrices and vectors, are multiplications, transpose, unfolding or matricization and factorizations (also named decompositions) [34, 13]. The most used tensor products are the Kronecker product denoted by $\otimes$, Khatri-Rao product denoted by $\odot$, Hadamard product denoted by $\circledast$, external product denoted by $\circ$ and n-mode denoted by $\times_n$.

In the rest of the article, we use the boldface Euler script letters to indicate a tensor $\boldsymbol{\mathcal{X}}$, boldface capital letters $\mathbf{M}$ for matrices, boldface lowercase letters to indicate a vector $\mathbf{v}$, and an element of the tensor or a scalar is noted in italic, for example x_{ijk} is ijk-i-th element of 3-order tensor $\boldsymbol{\mathcal{X}}$.

3.1 TDM's Data Model

In TDM, tensor dimensions and values are represented by associative arrays. In the general case, an associative array is a map from a key space to a value space and can be implemented using hash table or tree.

Definition 1 (Associative Array). *An associative array is a map that associates keys to values as* $A : K_1 \times \ldots \times K_N \rightarrow \mathbb{V}$ *where* $K_i, i = 1, .., N$ *are the sets of keys and* $\mathbb{V}$ *is the set of values.*

The definition given in [27] restricts $\mathbb{V}$ to have a semi-ring structure and the associative array to have a finite support. In TDM we use associative arrays in three different cases.

First, we use different associative arrays denoted by $\mathbf{A}_i$ for $i = 1, .., N$ to model dimensions of a tensor $\boldsymbol{\mathcal{X}}$, in this case the associative array has only one set of keys associated with integers $\mathbf{A}_i : K_i \rightarrow \mathbb{N}$ and $\mathbf{A}_i$ represents bijective functions. For example $\mathbf{A}_1 : String \rightarrow \mathbb{N}$ associates integers to users names. Their values are obtained by native queries sent to storage systems.

Second, at a lower level, an associative array can be used to represent the values of a sparse N-order tensor by associating compound keys from dimensions to values (real, integer) $\mathbf{A}_{vst} : K_1 \times \ldots \times K_N \rightarrow \mathbb{V}$.

Third, for tensors with non numerical values, two associative arrays are used as an indirection, one to map keys dimensions to a set integer keys ($\mathbf{A}_{vst}$) and another one to map the integer keys to non-numeric domains values (one integer is associated with each different value).

Definition 2 (Named Typed Associative Array). *A named and typed associative array of a tensor* $\boldsymbol{\mathcal{X}}$ *is a triple* $(Name, \mathbf{A}, T_A)$ *where* $Name$ *is a unique string which represents the name of a dimension,* $\mathbf{A}$ *is the associative array, and* T_A *the type of the associative array i.e.* $K \rightarrow \mathbb{N}$.

The signature of a named typed associative array is $Name : K \rightarrow \mathbb{N}$. The schema of a named typed associative array is $(Name : Dom_{\mathbf{A}})$, its set of key values is noted $Name.Keys$

Definition 3 (Typed Tensor). *A typed tensor* $\boldsymbol{\mathcal{X}}$ *is a tuple* $(Name, D_A, V, T)$ *where* $Name$ *is the name of the tensor,* D_A *is a list of named typed associative arrays i.e., one named typed associative array per dimension,* V *is an associative array that stores the values of the tensor and* T *is the type of the tensor, i.e. the type of its values.*

The schema of a tensor is the concatenation of $Name : T$ with the schema of all elements of D_A. For example if a tensor represents the hashtags of tweets published by a user during an hour, the schema will be $(UHT : \mathbb{N}, U : String, H : String, T : Integer)$. A TDM schema is a set of typed tensors schema.

If we consider the representation of tensor values, V handles the sparsity of tensors. Sparse tensors have a default value (e.g. 0) for all the entries that not explicitly exist in the associative array. Associative array refers to the general mathematical concept of map or function, as we want to conform to the separation of logical and physical levels, the associative arrays in the model are abstract data types that can be implemented using different representation techniques as well as for the tensor values. For example Kuang et al. [36] describe a unified tensor model to represent unstructured, semi-structured, and structured data

and propose a tensor extension operator to represent various types of data as sub-tensors merged into a unified tensor. Lara [25, 26] proposes a logical model and an algebra using associative array (called associative table) with a set of operations to unify different data models such as relational, array and key-value. The authors show how to use Lara as middleware algebra, their approach is directed towards operators translation and optimization. However their model is not very suitable for expressing high-level data transformations as tensors can do with their capacity of modeling complex relationships (i.e. not only binary).

3.2 Translating TDM's and other Models

In this section we establish mappings between TDM and other data models with the assumption that associative arrays are invariant to permuting keys.

1. a relation R is a set of tuples $(v_1, v_2, \ldots, v_k)$, where each element v_j is a member of a domain Dom_j, so the set-theoretic relation R is a subset of the cartesian product of the domain $Dom_1 \times Dom_2 \times \ldots \times Dom_k$. We can write a typed tensor $\boldsymbol{\mathcal{X}}$ using the name of each associative array in D_A as domain $Dom_i, i = 1, \ldots, N$ for R and by adding an attribute whose domain is the name of the tensor. The values of a tuple are those corresponding to keys of each D_A associated with the values of $\boldsymbol{\mathcal{X}}$. The names of D_A form a compound key for R. The reverse mapping from a relation R to typed tensors produces a set of tensors $\boldsymbol{\mathcal{X}}_i$ where the dimensions are the n attributes that are the key of R and for the $k - n$ remaining attributes we create a tensor for each. The keys of each D_A are formed of the different values of each attribute domains.
2. most of key-value stores save data as ordered $(key, value)$ pairs in a distributed hash table [7]. As typed tensor schema and its values are described by associative arrays there is a straight mappings between this type of NoSQL store and Tensor Data Model.
3. a column store system, like Vertica or Cassandra, uses a relational-like schema so their mapping to Tensor Data Model is the same as for relation.
4. a graph $G = (V, E)$, where V is the set of vertices and $E \subset V \times V$ the set of edges, can be represented by its adjacency or incidence matrices i.e. a 2-order tensor. Matrices can also represent oriented, weighted graphs. For multigraph, i.e. graph with different types of links for which $E = \{E_1, E_2, \ldots, E_k\}$ is partitioned set of edges, can be modeled by a 3-order tensor where one dimension is used to specify the different edges types. Moreover, multi-layer network [32] is defined as $GM = (V, E, L)$ where $V = \{V_1, V_2, \ldots, V_n\}$ is a partitioned set of nodes, $E = \{E_1, E_2, \ldots, E_k\}$ is partitioned set of edges, with $E \subseteq V \times V$ and $E_i \subseteq V_l \times V_m$ for $i \in \{1, \ldots, k\}$ and $l, m \in \{1, \ldots n\}$. L is a partitioned set of layers, $L = \{L_1, L_2, \ldots, L_p\}$ where $L_i \subseteq E$, with $L_i \cap L_j = \emptyset, \forall i, j$ modeling the dimensions. The construction of a tensor $(2(p + 1)$ order) for multi-layered networks is given in [14, 32]. Hypergraphs are also taken in consideration in [32] that shows their mapping to tensors. Graph databases handle in different way theoretical models of graphs [4]

most of them except maybe the nested-graph can be generalized by multi-layer graph and specified by tensors.

All the above models are structurally equivalent to TDM. The most appropriate storage system can be chosen based on the nature of the data and the cost models associated with the operations to be favored. Moreover, in specific cases part of data can be duplicated. We have studied some real examples of such equivalences in [37].

3.3 TDM's Operators

To carry out a wide range of queries it should possible to define several of the standard operators from relational algebra in terms of tensor operations. In [26, 27] the authors define a model and operators over associative arrays to unify relational, arrays, and key-value algebras. Our operators are defined to provide programmers with a logical data independence layer i.e. to bridge the semantic gap between analysis tools and storage systems. Our set of operators works on typed tensors at two different levels: at the associative array level and at the tensor value level. We focus on the following subset of operators on typed tensors: selection, projection, union, intersection, join and some analytic operators such as group by and tensors decomposition.

Data Manipulation Operators.

Projection operators are the usual operators of tensor algebra. A fiber of a tensor $\boldsymbol{\mathcal{X}}$ is a vector obtained by fixing all but one $\boldsymbol{\mathcal{X}}$'s indices: $\boldsymbol{\mathcal{X}}_{:jk}$, $\boldsymbol{\mathcal{X}}_{i:k}$ and $\boldsymbol{\mathcal{X}}_{ij:}$. Fibers are always assumed to be column vectors, similar to matrix rows and columns. A slice of a tensor $\boldsymbol{\mathcal{X}}$ is a matrix obtained by fixing all but two of $\boldsymbol{\mathcal{X}}$'s indices: $\boldsymbol{\mathcal{X}}_{i::}$, $\boldsymbol{\mathcal{X}}_{:j:}$ et $\boldsymbol{\mathcal{X}}_{::k}$. A project operator can be generalized by using the mode-n product $\times_n$ (mode-n product behavior is detailed in [34]).

Definition 4 (Project). *The projection of a N-order typed tensor $\boldsymbol{\mathcal{X}}$ on one or more mode, noted as $\Pi_{mode}(expr)$, where $mode = 1, \ldots, N$ and $expr$ is an equality between the name of an associative array and a constant value, reduces the dimensions of the tensor to the selected ones.*

The project operator can be computed by using the n-mode product with a boolean vector that contains 1 for the elements of the mode(s) to retain: $\boldsymbol{\mathcal{X}} \times_n \mathbf{b}$. For example, consider a 3-order tensor, $\boldsymbol{\mathcal{X}}_1$, with dimensions users, hashtags and time used to store the number of times a hashtag is used in tweets by a user per time slice. The number of each hashtag used by a user u_i, for all time slices is a 2-order tensor such as: $\boldsymbol{\mathcal{X}}_2 = \boldsymbol{\mathcal{X}}_1 \times_1 \mathbf{b}$ with $b_i = 1, b_j = 0, \forall j \neq i$.

A selection operator can be defined on two levels: 1) on the values contained in the tensor or 2) on the values of the dimensions i.e. typed associative arrays $\mathbf{A}_i$, $i = 1, .., N$.

Definition 5 (Select on tensor values). *The operator $\sigma_{expr}\boldsymbol{\mathcal{X}}$ selects values of the tensor which satisfy $expr$ and produces a new tensor with the same schema.*

The connectors allowed in expr are $\wedge$, $\vee$ *and* $\neg$, *the binary operators are* $\{<, \leq, =, \neq, \geq, >\}$. *The implicit left operand is a variable (values of* $\boldsymbol{\mathcal{X}}$*), the right one is a constant.*

For example $\sigma_{[>10]}\boldsymbol{\mathcal{X}}$ produces a new tensor (user, hashtag, time) with users that have published one or more tweets during time slice using more than 10 times the same hashtag.

Definition 6 (Restriction on dimensions values). *The operator* $\rho_{expr}\boldsymbol{\mathcal{X}}$ *restricts the tensor shape by selecting some values of the dimensions contained in the propositional formula expr. The connectors allowed in expr are* $\wedge$, $\vee$ *and* $\neg$, *the binary operators are* $\{<, \leq, =, \neq, \geq, >\}$, *the terms or variables are the sames of the associative arrays corresponding to dimension.*

The schema of a tensor is not affected by the restriction operator nor the schema of its typed associative arrays. The following example selects hashtags used by the user $u1$ for time slices between 18-03-08 and 18-02-28, from the tensor $\boldsymbol{\mathcal{X}}$:

$$\rho_{[U=\text{'u1'} \wedge T \geq \text{'18-02-28'} \wedge T \leq \text{'18-03-08'}]}\boldsymbol{\mathcal{X}}$$

Definition 7 (Union). *The union of two typed tensors* $\boldsymbol{\mathcal{X}}_1$ *and* $\boldsymbol{\mathcal{X}}_2$ *having the same schema, noted* $\cup_\theta$, *where* $\theta \in \{+, -, \times, \div, max, min\}$ *is a typed tensor* $\boldsymbol{\mathcal{X}}_3$ *with the same schema,* $\boldsymbol{\mathcal{X}}_3.D_{A_i}.Keys = \boldsymbol{\mathcal{X}}_1.D_{A_i}.Keys \cup \boldsymbol{\mathcal{X}}_2.D_{A_i}.Keys$, *for* $i = 1, \ldots, N$. *Values of* $\boldsymbol{\mathcal{X}}_3$ *are values from* $\boldsymbol{\mathcal{X}}_1$ *and* $\boldsymbol{\mathcal{X}}_2$ *except for keys in common for which the operator* θ *is applied.*

Definition 8 (Intersection). *The intersection of two typed tensors* $\boldsymbol{\mathcal{X}}_1$ *and* $\boldsymbol{\mathcal{X}}_2$ *having the same schema, noted* $\cap_\theta$, *where* $\theta \in \{+, -, \times, \div, max, min\}$ *is a typed tensor* $\boldsymbol{\mathcal{X}}_3$ *with the same schema,* $\boldsymbol{\mathcal{X}}_3.D_{A_i}.Keys = \boldsymbol{\mathcal{X}}_1.D_{A_i}.Keys \cap \boldsymbol{\mathcal{X}}_2.D_{A_i}.Keys$, *for* $i = 1, \ldots, N$. *Values of* $\boldsymbol{\mathcal{X}}_3$ *are values associated to common keys of each dimension on which the operator* θ *is applied.*

Definition 9 (Join). *The join of two typed tensors* $\boldsymbol{\mathcal{X}}_1$ *and* $\boldsymbol{\mathcal{X}}_2$ *having at least one common dimension, noted* $\bowtie_\theta$ *is a typed tensor* $\boldsymbol{\mathcal{X}}_3$ *which schema is the union of the two sets of dimension. The keys retained in the common dimensions are those satisfying the operator* θ, *where* $\theta \in \{<, \leq, =, \neq, \geq, >\}$.

Analytical Operators.
Group by like operations [33, 43] can be defined on typed tensors applying aggregation function with selection of tensor dimension values to aggregate.

Definition 10 (Aggregation). *The aggregation applied on a typed tensor* $\boldsymbol{\mathcal{X}}$ *is noted* $\mathcal{F}_{expr}(assoc)$ *where expr is a list of expressions of the* $op(name)$ *where* $op \in \{SUM, AVG, COUNT, MIN, MAX\}$ *and name is the name of an associative array in* D_A, *assoc is a list of names of associative arrays in* D_A. *The aggregation operator applies operators on specified dimensions for equals values of keys to produce a typed tensor with a schema specified by assoc.*

This operator is useful to transform typed tensor to time series for example to obtain the total number of each hashtag used during each time slice.

Tensor decompositions such as CANDECOMP/PARAFAC (CP), Tucker, HOSVD are used to perform dimensionality reductions and to extract latent relations [34]. Since tensor representations of data are multiple and their semantics are not explicit, the results of tensor decompositions are complex to interpret. Research in applied mathematics is important, it concerns the problems of robustness, remote calculations and value reconstruction [28, 8, 10]. To develop an in-database approach, the cost models of the manipulation and analysis tensors operators must be studied carefully in regards to sparsity.

4 Architecture and experiments

We validate our approach by a *proof-of-concept*, showing how TDM is used in a polyglot architecture with real data from a multi-disciplinary project (TEP 2017) involving collaborations with communication scientists. The main research objective of TEP 2017 is to study the dynamics of political discourse on Twitter during the French Presidential election in March-May 2017.

The architecture set up to carrying out the experimentation includes a polystore built on the top of PostgreSQL, HDFS, and Neo4j and three analysis frameworks R, Spark and TensorFlow. The data set is a tweets corpus captured during the period from 2017-02-27 to 2017-05-18.The data set contains 49 million tweets emitted by more than 1.8 million different users using 288, 221 different hashtags. Of the 49 million tweets, 36 million are retweets. Raw data are stored in JSON file format in HDFS (720Go), most important attributes of tweets are stored in a relational database (PostgreSQL) in a unique table (50Go), in another database with a normalized schema (55Go), links between entities (tweets, users, hashtags) are stored in Neo4j (23Go).

The abstraction layer is developed using R and Spark connectors. Associative arrays are implemented using RDD and dataframes. More details on the architecture (figure 2) are given in [37].

In order to study the possible influence of robots on the circulation of viral tweets, we sought to detect robots among Twitter accounts which had retweeted at least 1,000 times over the period between the two rounds of this election (from 23^{rd} April until 7^{th} May 2017), reducing the corpus of 49M of tweets to one thousand. In order to reduce the number of accounts to analyze, only accounts having tweeted more the 100 times in the 2-weeks period are retained. 1,077 accounts are selected in this way. This corresponds to the hypothesis that robots are tweeting intensively during these final period of the election campaign. A second hypothesis is that robots does not tweet in hazard so we extract hashtags contained in these tweets. We built a 3-order tensor modelling these accounts A, the hashtags H and the time T (2-weeks period). We got a tensor containing potentially $1,077 \times 568 \times 336$ items. The tensor construction from the relational data requires approximately 15 SQL lines and less of 5 minutes including a few seconds for the associative arrays. TensorFlow is used as reference for simulating

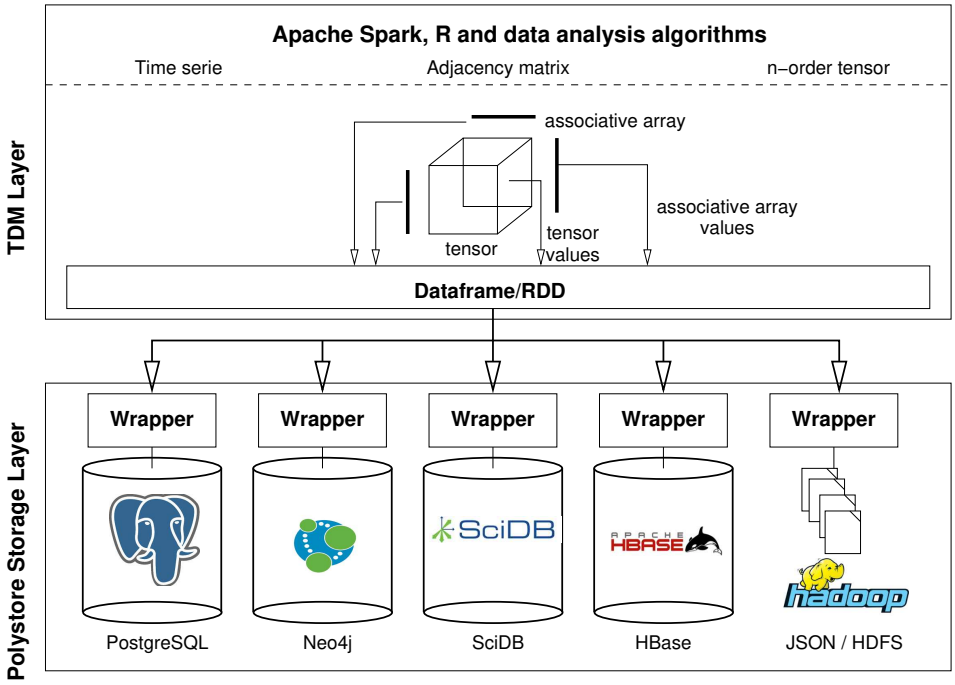

Fig. 2. Architecture of the polyglot system using TDM

a traditional framework without a polystore connection using data exchange with file and data transformation programs. Around a hundred lines of complex SQL queries are required to produce a tensor for TensorFlow.

We performed a CP decomposition to reduce the user space based on their behavior, it produces n groups of three 1-order tensors, here vectors A, H, T. We then apply the k-means clustering algorithm to identify groups of users having similar behavior in the time. A k-means algorithm applied to this data determines 4 groups of users: a group of one account previously detected as a robot and suspended by Twitter, a group of three accounts, a group of about thirty accounts and a last group containing other users. The group of three accounts, revealed after manual study, to be linked (same behavior and hashtags) and assisted by an algorithm that retweets messages against the Macron candidate. Each tweeted between 1,000 and 1,800 times during the period with a vast majority of retweets.

We also used the Louvain algorithm [11] to detect accounts which retweet or are retweeted frequently by the other accounts and which tend to share the same retweets. The retweet graph represented by its adjacency matrix is obtained from a 3-order tensor $(AAT : \mathbb{N}, A : String, A : String, T : Integer)$. The result of the CP decomposition is confirmed (figure 3): yellow community corresponds to the user detected as robot, pink community corresponds to the group of three users

and others (blue and green communities) are the users of the third group. The biggest nodes are accounts from the 4 clusters obtained by the CP decomposition and k-means.

Fig. 3. Communities obtained from retweet graph

Our result is consistent with the use of OSoMe API Botometer[18] which provides an overall probability score that an account studied is automated. It uses 1,150 criteria and machine learning techniques to calculate the probability [51]. The group of about thirty accounts comprises more than half users with a probability of being a robot greater than 0.42 (figure 4). But we note that the values of the probabilities were not enough significant to detect robots during the studied period. One of the assumptions is that it is hybrid accounts of users assisted by algorithms. However, simple criteria such as the maximum number of tweets published in one hour make it possible to unambiguously find some accounts with automated behavior, confirmed by the manual study.

[18] `https://botometer.iuni.iu.edu/`

from_user	user_id	nbrt	nbrtjour	nbtweetp2	proba	retrycount	maxrateh	htnb	cluster
Manuel_Hollande	639	1073	76	1078	0.66	1	113	155	1
Macron_Jamais	622	1481	105	1706	0.61	1	88	242	1
Degage_Hollande	277	1394	99	1464	0.66	0	118	189	1
[redacted]	724	1594	113	5688	0.5	0	188	276	3
[redacted]	933	3405	243	3727	0.31	1	149	512	4
[redacted]	477	2037	145	2370	0.39	0	96	437	4
[redacted]	761	2254	161	2263	0.3	1	93	381	4
[redacted]	1071	2289	163	2538	0.39	2	153	387	4
[redacted]	197	1988	142	2415	0.37	0	75	274	4
[redacted]	932	1967	140	2041	0.35	0	90	372	4
[redacted]	158	1932	138	1951	0.38	0	149	308	4
[redacted]	1070	1814	129	1820	0.54	0	174	177	4
[redacted]	979	1472	105	2396		9	133	236	4
[redacted]	712	1365	97	1428	0.48	0	131	231	4
[redacted]	968	4513	322	4558	0.39	1	214	590	4
[redacted]	248	6466	461	6672	0.5	0	141	869	4
[redacted]	345	6134	438	6549	0.34	0	242	911	4
[redacted]	446	5228	373	5336	0.34	0	120	722	4
[redacted]	112	1291	92	1395	0.69	0	95	125	4
[redacted]	834	7001	500	7105		8	106	1090	4
[redacted]	877	1223	87	1243	0.42	4	114	241	4
[redacted]	324	1108	79	1111	0.4	0	147	108	4
[redacted]	347	3525	251	3908	0.49	0	79	136	4
[redacted]	896	1509	107	1509	0.62	0	201	212	4
[redacted]	191	3337	238	3361	0.27	0	129	505	4
[redacted]	172	2917	208	2954	0.3	0	140	459	4
[redacted]	644	3431	245	3431	0.42	1	124	285	4
07_mai_2017	2	7233	516	7233		8	516	1822	6

(28 rows)

Fig. 4. Results given by Botometer (column with label proba) and cluster number obtained by the CP decomposition (column with label cluster)

5 Conclusion

In this article we described a Tensor Data Model for polystore systems and studied its ability to generalized several kinds of models such as relational, column, key-value, graph (including multi-layer graphs). Using associative array we defined schema, manipulation and analytics operators. We are working on an implementation of each operator in Spark and R.

We defined a set of real experiments with Twitter data to evaluate the ease of use of the operator toolkit and the performance of the architecture. We detected the possible influence of robots on the circulation of viral tweets. Our results have been validated by researchers in communication science. The experiments demonstrated the TDM capabilities according to the ease of data transformations in analyses.

Our short term future work will be on the storage of tensor as materialized views in SciDB through a matricization process [31]. As there is multiple ways of doing matricization, one must be chosen according to the privileged operators and it should be necessary to specify normal forms to guide matricization.

References

1. Abo Khamis, M., Ngo, H.Q., Nguyen, X., Olteanu, D., Schleich, M.: In-database learning with sparse tensors. In: Proceedings of the 35th ACM SIGMOD/PODS Symposium on Principles of Database Systems. pp. 325–340. ACM (2018)
2. Abouzeid, A., Bajda-Pawlikowski, K., Abadi, D., Silberschatz, A., Rasin, A.: Hadoopdb: an architectural hybrid of mapreduce and dbms technologies for analytical workloads. Proceedings of the VLDB Endowment **2**(1), 922–933 (2009)
3. Allen, D., Hodler, A.: Weave together graph and relational data in apache spark. In: Spark+AI Summit. Neo4j (2018), `https://vimeo.com/274433801`

4. Angles, R.: A comparison of current graph database models. In: Data Engineering Workshops (ICDEW), 2012 IEEE 28th International Conference on. pp. 171–177. IEEE (2012)
5. Arora, S., Ge, R., Moitra, A.: Learning topic models–going beyond svd. In: Foundations of Computer Science (FOCS), 2012 IEEE 53rd Annual Symposium on. pp. 1–10. IEEE (2012)
6. Astrahan, M.M., Blasgen, M.W., Chamberlin, D.D., Eswaran, K.P., Gray, J.N., Griffiths, P.P., King, W.F., Lorie, R.A., McJones, P.R., Mehl, J.W., et al.: System R: relational approach to database management. ACM Transactions on Database Systems (TODS) **1**(2), 97–137 (1976)
7. Atikoglu, B., Xu, Y., Frachtenberg, E., Jiang, S., Paleczny, M.: Workload analysis of a large-scale key-value store. In: ACM SIGMETRICS Performance Evaluation Review. vol. 40, pp. 53–64. ACM (2012)
8. Austin, W., Ballard, G., Kolda, T.G.: Parallel tensor compression for large-scale scientific data. In: Parallel and Distributed Processing Symposium, 2016 IEEE International. pp. 912–922. IEEE (2016)
9. Baazizi, M.A., Lahmar, H.B., Colazzo, D., Ghelli, G., Sartiani, C.: Schema inference for massive json datasets. In: Extending Database Technology (EDBT). pp. 222,233 (2017)
10. Battaglino, C., Ballard, G., Kolda, T.G.: A practical randomized CP tensor decomposition. arXiv preprint arXiv:1701.06600 (2017)
11. Blondel, V.D., Guillaume, J.L., Lambiotte, R., Lefebvre, E.: Fast unfolding of communities in large networks. Journal of Statistical Mechanics: Theory and Experiment **2008**(10), P10008 (2008), `http://stacks.iop.org/1742-5468/2008/i=10/a=P10008`
12. Brodie, M.L., Schmidt, J.W.: Final report of the ansi/x3/sparc dbs-sg relational database task group. ACM SIGMOD Record **12**(4), 1–62 (1982)
13. Cichocki, A., Zdunek, R., Phan, A.H., Amari, S.: Nonnegative matrix and tensor factorizations: applications to exploratory multi-way data analysis and blind source separation. John Wiley & Sons (2009)
14. De Domenico, M., Solé-Ribalta, A., Cozzo, E., Kivelä, M., Moreno, Y., Porter, M.A., Gómez, S., Arenas, A.: Mathematical formulation of multilayer networks. Physical Review X **3**(4), 041022 (2013)
15. DiScala, M., Abadi, D.J.: Automatic generation of normalized relational schemas from nested key-value data. In: Proceedings of the 2016 International Conference on Management of Data. pp. 295–310. ACM (2016)
16. Duggan, J., Elmore, A.J., Stonebraker, M., Balazinska, M., Howe, B., Kepner, J., Madden, S., Maier, D., Mattson, T., Zdonik, S.: The BigDAWG Polystore System. ACM SIGMOD Record **44**(2), 11–16 (2015)
17. Franklin, M., Halevy, A., Maier, D.: From databases to dataspaces: a new abstraction for information management. ACM SIGMOD Record **34**(4), 27–33 (2005)
18. Gadepally, V., Chen, P., Duggan, J., Elmore, A., Haynes, B., Kepner, J., Madden, S., Mattson, T., Stonebraker, M.: The BigDAWG polystore system and architecture. In: IEEE High Performance Extreme Computing Conference (HPEC). pp. 1–6 (2016)
19. Ghosh, D.: Multiparadigm Data Storage for Enterprise Applications. IEEE Software **27**(5), 57–60 (2010)
20. Haerder, T., Reuter, A.: Principles of transaction-oriented database recovery. ACM Computing Surveys (CSUR) **15**(4), 287–317 (1983)
21. Härder, T.: Dbms architecture-the layer model and its evolution. Datenbank-Spektrum **13**, 45–57 (2005)

22. Hellerstein, J.M., Ré, C., Schoppmann, F., Wang, D.Z., Fratkin, E., Gorajek, A., Ng, K.S., Welton, C., Feng, X., Li, K., et al.: The madlib analytics library: or mad skills, the sql. Proceedings of the VLDB Endowment **5**(12), 1700–1711 (2012)
23. Hogben, L.: Handbook of linear algebra. Chapman and Hall/CRC (2013)
24. Hölsch, J., Schmidt, T., Grossniklaus, M.: On the performance of analytical and pattern matching graph queries in Neo4j and a relational database. In: EDBT/ICDT 2017 Joint Conference: 6th International Workshop on Querying Graph Structured Data (GraphQ) (2017)
25. Hutchison, D., Howe, B., Suciu, D.: Lara: A key-value algebra underlying arrays and relations. arXiv preprint arXiv:1604.03607 (2016)
26. Hutchison, D., Howe, B., Suciu, D.: LaraDB: A minimalist kernel for linear and relational algebra computation. In: Proceedings of the 4th ACM SIGMOD Workshop on Algorithms and Systems for MapReduce and Beyond. pp. 2–12. ACM (2017)
27. Jananthan, H., Zhou, Z., Gadepally, V., Hutchison, D., Kim, S., Kepner, J.: Polystore mathematics of relational algebra. In: IEEE International Conference on Big Data (Big Data). pp. 3180–3189 (Dec 2017). https://doi.org/10.1109/BigData.2017.8258298
28. Kang, U., Papalexakis, E., Harpale, A., Faloutsos, C.: Gigatensor: Scaling tensor analysis up by 100 times - algorithms and discoveries. In: Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 316–324. KDD '12, ACM (2012)
29. Kepner, J., Arcand, W., Bergeron, W., Bliss, N., Bond, R., Byun, C., Condon, G., Gregson, K., Hubbell, M., Kurz, J., et al.: Dynamic distributed dimensional data model (D4M) database and computation system. In: IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). pp. 5349–5352. IEEE (2012)
30. Kepner, J., Arcand, W., Bestor, D., Bergeron, B., Byun, C., Gadepally, V., Hubbell, M., Michaleas, P., Mullen, J., Prout, A., et al.: Achieving 100,000,000 database inserts per second using Accumulo and D4M. In: High Performance Extreme Computing Conference (HPEC). pp. 1–6. IEEE (2014)
31. Kim, M.: Tensordb and tensor-relational model (trm) for efficient tensor-relational operations (2014)
32. Kivelä, M., Arenas, A., Barthelemy, M., Gleeson, J.P., Moreno, Y., Porter, M.A.: Multilayer networks. Journal of Complex Networks **2**(3), 203–271 (2014)
33. Klug, A.: Equivalence of relational algebra and relational calculus query languages having aggregate functions. J. ACM **29**(3), 699–717 (Jul 1982)
34. Kolda, T.G., Bader, B.W.: Tensor decompositions and applications. SIAM review **51**(3), 455–500 (2009)
35. Kolev, B., Bondiombouy, C., Valduriez, P., Jiménez-Peris, R., Pau, R., Pereira, J.: The CloudMdsQL Multistore System. In: Proceedings of the International Conference on Management of Data (SIGMOD). pp. 2113–2116 (2016)
36. Kuang, L., Hao, F., Yang, L.T., Lin, M., Luo, C., Min, G.: A tensor-based approach for big data representation and dimensionality reduction. IEEE transactions on emerging topics in computing **2**(3), 280–291 (2014)
37. Leclercq, E., Savonnet, M.: A Tensor Based Data Model for Polystore: An Application to Social Networks Data. In: Proceedings of the 22nd International Database Engineering & Applications Symposium (IDEAS). pp. 1–9. ACM, New York, NY, USA (2018)
38. Leskovec, J., Rajaraman, A., Ullman, J.D.: Mining of massive datasets. Cambridge university press (2014)

39. Li, X., Cui, B., Chen, Y., Wu, W., Zhang, C.: Mlog: towards declarative in-database machine learning. Proceedings of the VLDB Endowment **10**(12), 1933–1936 (2017)
40. Litwin, W., Abdellatif, A., Zeroual, A., Nicolas, B., Vigier, P.: Msql: A multidatabase language. Information sciences **49**(1-3), 59–101 (1989)
41. Ong, K.W., Papakonstantinou, Y., Vernoux, R.: The SQL++ unifying semi-structured query language, and an expressiveness benchmark of SQL-on-Hadoop, NoSQL and NewSQL databases. Tech. rep., UCSD (2014)
42. Ong, K.W., Papakonstantinou, Y., Vernoux, R.: The SQL++ Query Language: Configurable, Unifying and Semi-structured. Tech. rep., UCSD (2015)
43. Özsoyoğlu, G., Özsoyoğlu, Z.M., Matos, V.: Extending relational algebra and relational calculus with set-valued attributes and aggregate functions. ACM Trans. Database Syst. **12**(4), 566–592 (Nov 1987)
44. Özsu, M.T., Valduriez, P.: Principles of distributed database systems. Springer Science & Business Media (2011)
45. Sharp, J., McMurtry, D., Oakley, A., Subramanian, M., Zhang, H.: Data Access for Highly-Scalable Solutions: Using SQL, NoSQL, and Polyglot Persistence. Microsoft patterns & practices, 1st edn. (2013)
46. Singh, D., Reddy, C.K.: A survey on platforms for big data analytics. Journal of Big Data **2**(1), 8 (2015)
47. Stonebraker, M., Bear, C., Çetintemel, U., Cherniack, M., Ge, T., Hachem, N., Harizopoulos, S., Lifter, J., Rogers, J., Zdonik, S.: One size fits all? part 2: Benchmarking results. In: Proc. CIDR (2007)
48. Stonebraker, M., Cetintemel, U.: " one size fits all": an idea whose time has come and gone. In: Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on. pp. 2–11. IEEE (2005)
49. Tan, R., Chirkova, R., Gadepally, V., Mattson, T.G.: Enabling query processing across heterogeneous data models: A survey. In: IEEE International Conference on Big Data (Big Data). pp. 3211–3220. IEEE (2017)
50. Vargas-Solar, G., Zechinelli-Martini, J.L., Espinosa-Oviedo, J.A.: Big data management: What to keep from the past to face future challenges? Data Science and Engineering **2**(4), 328–345 (2017)
51. Varol, O., Ferrara, E., Davis, C.A., Menczer, F., Flammini, A.: Online Human-Bot Interactions: Detection, Estimation, and Characterization. In: Proceedings of the Eleventh International Conference on Web and Social Media (ICWSM). pp. 280–289 (2017)
52. Wang, J., Baker, T., Balazinska, M., Halperin, D., Haynes, B., Howe, B., Hutchison, D., Jain, S., Maas, R., Mehta, P., et al.: The Myria big data management and analytics system and cloud services. In: CIDR (2017)
53. Zhou, Y., Wilkinson, D., Schreiber, R., Pan, R.: Large-scale parallel collaborative filtering for the netflix prize. In: International Conference on Algorithmic Applications in Management. pp. 337–348. Springer (2008)