Data Virtual Machines: Data-Driven Conceptual Modeling of Big Data Infrastructures

Damianos Chatziantoniou
Athens University of Economics and Business
damianos@aueb.gr

Verena Kantere
National Technical University of Athens
verena@dblab.ece.ntua.gr

CEUR-WS.org/Vol-2578/SEAData4.pdf

ABSTRACT

In this paper we introduce the concept of Data Virtual Machines (DVM), a graph-based conceptual model of the data infrastructure of an organization, much like the traditional Entity-Relationship Model (ER). However, while ER uses a *top-down* approach, in which real-world entities and their relationships are depicted and utilized in the production of a relational representation, DVMs are based on a *bottom up* approach, mapping the data infrastructure of an organization to a graph-based model. With the term "data infrastructure" we refer to not only data persistently stored in data management systems adhering to some data model, but also of generic data processing tasks that produce an output useful in decision making. For example, a python program that "does something" and computes for each customer her probability to churn is an essential component of the organization's data landscape and has to be made available to the user, e.g. a data scientist, in an easy to understand and intuitive to use manner, the same way the age or gender of a customer are made. In fact, a DVM depicts only associations between attributes (nodes). An association is given by some computation on the underlying data that relates values of these attributes. In this respect, is model-agnostic. However, a DVM can be quite helpful in a variety of crucial tasks carried out by statisticians and data engineers.

KEYWORDS

data virtualization, data virtual machines, big data infrastructures

1 INTRODUCTION

Modern organizations collect, store and analyze a wealth of data from different sources and applications, used in a variety of data analysis projects, such as traditional BI, data exploration, data mining, etc. to provide a competitive advantage to the business. This data has to be integrated to provide the data scientist with a "holistic" view of the enterprise's data infrastructure. The term *data infrastructure* encompasses much more than data persistently stored in data management systems. It also involves processes that can be useful during analysis, such as a Python program that computes the social influence of each customer.

In a recent project at a major telecom provider, we had to predict churn in the presence of structured and unstructured data residing at different systems, relational and non-relational. For this project a predictive model had to be designed and implemented taking into account the many possible variables (features) characterizing the customer (demographic, interactions with call center, emails, social data, etc.) The goal was to equip the data scientist with a simple tool that would allow her to choose and experiment in an ad-hoc manner with multiple tabular views of customer-related data. We wanted to create a "virtual data desktop", where schema designers (IT people) could rapidly map customer's attributes, and data scientists could simply define (possibly in a polyglot manner) transformations over attributes and combine them into dataframes. The evaluation of dataframes should be efficient and based on a solid theoretical framework.

Data integration can be seen as constructing a data warehouse, or creating a virtual database [6]. It is worth mentioning that defining global views over heterogeneous data sources is not a big data-era issue and has been extensively discussed in the past (e.g. [1]). While data warehousing was the way to go in the past – mainly due to the dominance of relational systems in data management – there are well-thought arguments to reconsider a virtual database approach, a rapidly emerging trend in the business world as data virtualization [9]. There are needs to accommodate data regulations; manage schema in an agile manner; integrate rapidly (parts of) data sources; perform ad hoc data preparation without the rigidity of data warehouses. All these requirements can be served well with a virtual approach. The focus of this era is on "schematic flexibility/versatility" rather than "querying performance". Similar arguments can be found in [10], proposing a data virtualization approach and in Polystores [7].

In this paper we discuss our vision for the creation of a data virtual machine, as a graph-based conceptual model which is built bottom-up. The high level goals of this work are:

Support for end-to-end processing This is a well-known research mandate for the data analysis pipeline [8], stating the need for the "development of multiple tools, each solving some piece of the raw-data-to-knowledge puzzle, which can be seamlessly integrated and be easy to use for both lay and expert users." Seamless integration requires a high-level conceptual layer where data, processes and models can be easily mapped and manipulated.

Coping with diversity in the data management landscape: This is another well-known research mandate [4], [8]. Multiple big data systems and analysis platforms need to coexist, and query support that span such systems necessitates that platforms are integrated and federated. While data warehousing is the de facto approach, it is rigid for a rapidly changing data environment.

Redefining data infrastructure: An organization's data infrastructure includes data stored persistently (possibly adhering to different data models) but also programs that produce output useful in the analysis phase. Both data and programs should be treated as first class citizens and both should be mapped in a high-level conceptual model.

Polyglotism: A data scientist can choose from a range of functions/methods in different programming languages to perform a specific task (e.g. extraction of sentiment, scoring, aggregation, etc.) She should be enabled to use these in the same query and the query evaluation engine should handle them efficiently.

Visual Schema Management and Query Formulation Data scientists do not necessarily understand relational modeling or know SQL, which both could become quite complex for large

© 2020 Copyright for this paper by its author(s). Published in the Workshop Proceedings of the EDBT/ICDT 2020 Joint Conference (March 30-April 2, 2020, Copenhagen, Denmark) on CEUR-WS.org. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0)

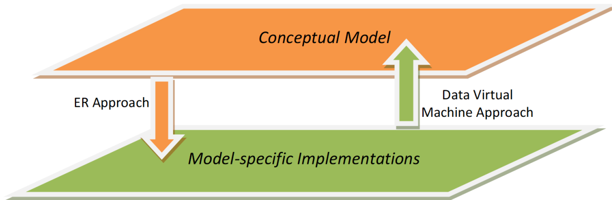


Figure 1: DVM-modeling vs traditional ER-modeling

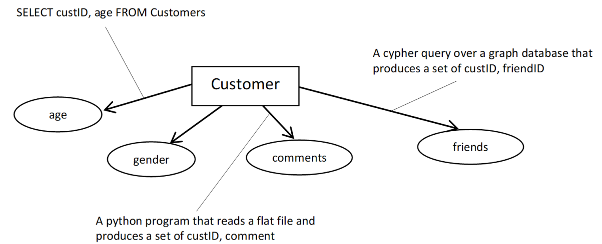


Figure 2: A customer entity with several attributes

schemas. They want to see visualizations that they can understand and explore, and they need to visually define transformations over attributes (variables); they need to navigate the schema to choose inputs for learning algorithms; and they want to easily extend the schema using wizards. Such needs lead to the requirement of a model that natively supports visual exploration and schema management and is amenable to the development of graphical query languages. Graph-based models are amenable to GUI implementations – more than other models.
Shareable: The model or parts of it must be easily available to third parties within or outside the enterprise, since the value of data explodes when the latter can be linked with other data [8].
Crawlable: Feature selection is a well-known process in statistics. The data scientist selects and places attributes in a dataframe. We need an automated way to generate such attributes related to specific entities. For this, we need a model that supports crawling, e.g. starting from an entity, an algorithm collects or defines relevant attributes. The web is an example of a crawlable model.

2 DATA VIRTUAL MACHINES

A Data Virtual Machine describes entities and their attributes in a graphical way, much like the traditional Entity-Relationship Model (ER). A conceptual model, like the ER is simple to understand, succinct, and depicts entities at a higher level. However, developing a conceptual model as a DVM is the reverse process of the one followed in a traditional ER design: while ER uses a top-down approach, DVM uses a bottom up approach, from existing data – stored in model-specific implementations – back to a conceptual model[1]. Figure 1 shows DVM- vs ER-modeling.

In the past there has been some little interest in the creation of bottom-up approaches for the construction of a RDF graph from the data. Some of these focus on the creation of RDF views on top of relational data, e.g. [12]. In this case, there is already a user-defined mapping between the relational schema and a target ontology, which is employed for the creation of a representation of relational schema concepts in terms of RDF classes and properties. The work in [11] also considers the same problem, i.e. given a relational database schema and its integrity constraints, a mapping to an OWL ontology is produced, which, provides the basis for generating RDF instances. Such works are orthogonal to our vision, as they assume that the starting point is a given relational schema, and the goal is to translate this schema into RDF (via the employment of ontologies). The notion of DVM that we discuss focuses on, first, creating an integrated conceptual model that can accommodate various data models, and, second, produce the conceptual schema based on the processing of the data, rather than the data itself.

The key idea in a DVM is to make it easy to add an entity or an attribute to an entity from a variety of data sources (relational databases, flat files, excel files, NoSQL, etc.) For instance, for a customer entity, examples of attributes include his age, gender and income, but also his emails, images, locations and transactions. An attribute of an entity could have one or more values – for example, the age of a customer is a single value, but the emails of a customer can be many – in ER theory these are called *multi-valued* attributes. In addition, attributes can be *derived*. A derived attribute is an attribute where its value is produced by some computational process, i.e. there exists a process that maps one or more values to the entity. In a DVM, since we map *existing* data to entities, we can only have derived attributes. For example, the query "`SELECT custID, age FROM Customers`" can be used to bind an age to the customer entity (using the primary key of the entity, custID). The computational process that "defines" the attribute (in this case, the SQL statement) accompanies, as semantics, the edge connecting the entity and the attribute. In this way, one can *semantically* represent *any* data processing task onto the conceptual model. Examples involve the SQL statement mentioned above, but also a MongoDB query, a Cypher query, programs that read from a flat or an excel file, even programs that assign a churn probability to a customer. The only requirement is that the data process maps one or more values to an entity, i.e. to have a two-column (id, value) output. An important observation to make is that this computation can be intra- or inter-organization. Figure 2 shows additional attributes for the customer entity (for simplicity we draw attributes with a solid line rather than a dashed line, as in traditional ER design). Let us assume that all entities have a primary key (a quite realistic assumption in most real-life implementations), so an entity (rectangle) can be represented by its primary key, which is also an attribute. In Figure 3, the customer entity is represented by the `custID` attribute. The transactions of a customer (consisting of `transIDs`) is another attribute (multi-valued) of the entity customer, but at the same time is an entity itself, with its own set of attributes, which means that there is no need for relationships, as in the traditional ER theory. This is also shown in Figure 3.

Finally, let us consider once again the query "`SELECT custID, age FROM Customers`". While this query maps an age to a custID, it also maps one or more custIDs to a specific age value. In other words, a data processing task with an output $\{(u, v) : u \in U, v \in V\}$ (multi-set semantics) provides two mappings, one from U to V and one from V to U. This means that edges in a DVM graph are bidirectional (Figure 3). In that respect, all nodes in this graph are equal, i.e. there is no hierarchy, and all connections are symmetrical, i.e. there are no primary keys. However, one can consider a node with degree > 1 as a "primary" key, shown in different color. A data virtual machine is as a graph-based arrangement of data processing tasks with output a pair of values, namely mappings between two attribute domains.

Definition 2.1 (Key-list Structure). A key-list structure K is a set of (key, list) pairs, $K = \{(k, L_k)\}$, where L_k is a list of elements

[1]We note that the conceptual model that the DVM follows is not the ER model, but an ER-like model, i.e. a model based on notions of entities and attributes. As such, it can also be characterized as an RDF-like model or a network-like model. For simplicity, in this paper, we make references to the ER model only

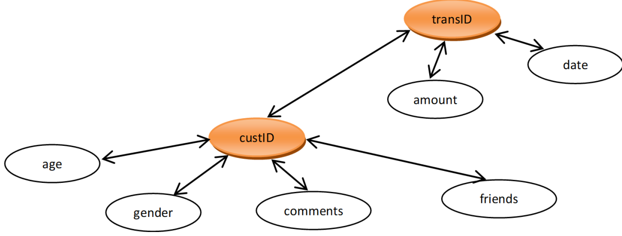
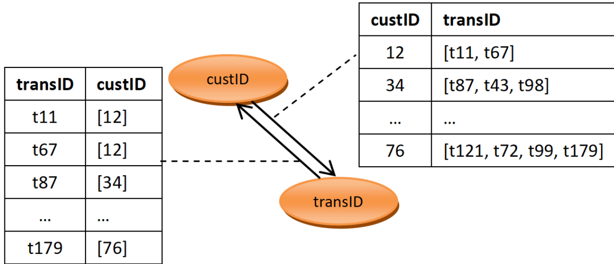


Figure 3: A simple DVM example



transID	custID
t11	[12]
t67	[12]
t87	[34]
...	...
t179	[76]

custID	transID
12	[t11, t67]
34	[t87, t43, t98]
...	...
76	[t121, t72, t99, t179]

Figure 4: Key-list structures to represent edges of DVMs

or the special value *null* and $\forall (k_1, L_{k1}), (k_2, L_{k2}) \in K$, $k_1 \neq k_2$. Both keys and elements of the lists are strings. □

Definition 2.2 (Data Virtual Machines). A Data Virtual Machine (DVM) is a (multi)graph that is constructed as follows:

- Assume n attributes $A_1, A_2, \ldots, A_n$ drawn from domains $D_1, D_2, \ldots, D_n$ respectively. Each attribute becomes a node in the graph.
- Assume one or more data processing tasks (DPT), where each DPT P has as output a multiset $S = \{(u, v) : u \in D_i, v \in D_j\}$. Based on S, one can define two key-list structures, denoted as $KL_{ij}(S)$ and $KL_{ji}(S)$ as:
  $K = \{k : (k, v) \in S\}$ (a set),
  $\forall k \in K, L_k = [v : (k, v) \in S]$, (a list),
  $KL_{ij}(S) = \{(k, L_k) : k \in K\}$

  $KL_{ji}(S)$ is similarly defined, treating as key the second constituent of the value pairs of S. For P we define two edges $A_i \longrightarrow A_j$ and $A_j \longrightarrow A_i$, each labeled with $KL_{ij}(S)$ and $KL_{ji}(S)$ respectively. □

Example 2.3. Assume the SQL query "`SELECT custID, transID FROM Customers` that maps transactions to customers and vice versa. The attributes, edges and the respective key-list structures are shown in Figure 4. □

The next section discussed that the concept of DVM is an appropriate high-level model for a big data environment.

3 CHALLENGES AND OPPORTUNITIES

DVMs allow the agile construction of graph-based schemas of existing data within an organization. We discuss below how DVMs contribute to the high level goals as set in Section 1

3.1 Model-agnostic data Sharing and Exchange

Any computation that produces as output a collection of pairs (programs, queries, web services), can be represented in a data virtual machine as an edge between two nodes. In real-world environments people need to share *parts of* spreadsheets, flat files, json documents or relations, and usually specific columns of these. This involves some excel formulas, db columns, or flat

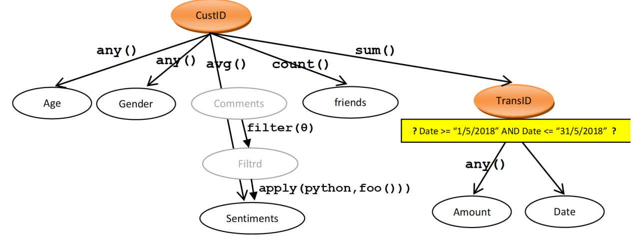


Figure 5: Visual representation of a dataframe query

files fields. In most cases this is done manually, by exporting to a csv and moving the file around. There is no principled way to describe formally these in an intermediate representation. DVMs can become the medium for data sharing in a standardized, collaborative, distributed manner. For example, a data engineer can select a pair of columns in an excel file and represent them as nodes in the DVM, via some wizard that generates the necessary computation. The data scientist can then use these within a data model. This is very similar to what ETL tools/Visualization tools do, when the output of the ETL task is binary (i.e. two columns).

3.2 Visual Query Formulation - Dataframes

What kind of queries can we have on top of DVMs? There is a large number of research papers on visual query formulation over ERs, dating back from the 80s, that are relevant here. But, let us consider what data scientists usually do, since this is the target group of this work. They usually form dataframes in Python, R or Spark. A dataframe is a table that is built incrementally, column-by-column. The first column(s) is some kind of key (customer ID, transaction ID, etc.) and the remaining ones are "attached" to the dataframe via a left-outer join on the key, denoting related "attributes". These columns may come from different data sources and can be transformed before being "glued" to the dataframe. A dataframe provides a tabular representation of an entity and usually serves as input to ML algorithms. We have extensively discussed this class of queries (termed as multi-feature queries, not dataframes), both in terms of syntax (by proposing SQL extensions [2]) and evaluation (by proposing a relational operator relying in parallel processing and in-memory evaluation techniques [3]). It is important to facilitate this process in a simple and intuitive, visual, manner.

One can easily pick and transform attributes (possibly along a path), to form a dataframe over a DVM. For example, using the DVM of Figure 2, one may want to form a dataframe using the `custID` as key (1st column) and her `age`, `gender`, the average sentiment of her `comments` containing the keyword "google", the count of her `friends` and the total `amount` of her transactions on May 2019, as additional columns. Graphically, the user selects a node as the key of the dataframe and one or more of that node's descendants as the additional columns. Aggregation is used to reduce multi-valued nodes to a single value. Figure 5 shows a visual representation of this query. The research questions focus on (i) what kind of dataframe queries one can express on top of a DVM, (ii) visual interfaces, and (iii) how can a system efficiently evaluate these queries. Regarding (i), there exists a well-defined grammar that specifies what is allowed and what is not (omitted here). In terms of efficient evaluation, a dataframe query is a tree rooted at a node of DVM. Recall from Section 2 that edges in a DVM correspond to key-list structures. One can define a set of operators having as input and output key-list structures, forming thus an algebra over key-list structures. For example, a filtering

operator could get a key-list structure and a condition θ and filter the list of each key based on this expression, producing a new key-list structure. Another operator could get as input a key-list structure and a function and apply it on each member of each list (mapping). A dataframing operator gets two or more key-list structures and join them on the key, unioning the matching keys' lists. As a result, dataframe queries can be expressed, optimized and evaluated within an algebraic framework.

3.3 Polyglot Data Preparation

The dataframe example of Section 3.2 involves a function written in some programming language (Python) to compute the sentiment of each comment. For a different attribute transformation in the *same* query, we can use a function in R. Finally, a user-defined aggregate function can be in another programming language. The query evaluation engine should support this kind of polyglotism within the algebraic framework. For example, while key-list structures could materialize within a single key-value system, the set of operators manipulating these structures could be implemented (redundantly) in different programming languages (i.e. all operators could be implemented both in R and Python and the query engine selects the PL-specific version of the operator(s) to apply, depending on the used function).

3.4 Accommodating Data Regulations

The EU General Data Protection Regulation (GDPR) driven by privacy concerns dictates that the data generated by the activity of an individual using a service can be deleted or exploited by the individual. Thus, one can ask the service owner to hand in all of her data. For example, a user could request from Facebook, Google or Wal-Mart for her activity data. One question is in what format these data will be handed to her, and another, how the user will create her *data portfolio*, i.e how she will represent and integrate these data, i.e. in which model: Relational? Semi-structured? Graph-based? Some sort of self-service data integration is necessary. The DVM model-agnostic exchange and integration capability can naturally serve this cause. The last question is on what the user can do with these data. Can she give them to a credit bureau to provide a specific evaluation on her? People already discuss micro-evaluation services on specific datasets. Also, she could just sell them. For this, the data model (or part of it) has to be shareable, e.g. available by a link. DVM seems as a good candidate to model, represent and share personal data. It is a graph-based conceptual model, focused on entities and attributes. Given a single entity, people easily understand the concept of an attribute: my age, my emails, my transactions, etc. A conceptual model also makes visualization easier and thus appropriate for some kind of *self-service data integration.*

3.5 Data Virtualization

Data virtualization is a relatively new business trend [5]. Companies like Denodo, Oracle, SAS and others already offer relevant products. Data virtualization is closely related to mediators and virtual databases, if not a reinvention of these. According to Wikipedia, "data virtualization is any approach to data management that allows an application to retrieve and manipulate data without requiring technical details about the data, such as how it is formatted at source, or where it is physically located, and can provide a single customer view (or single view of any other entity) of the overall data. Data virtualization may also be considered as an alternative to ETL and data warehousing. It is inherently aimed at producing quick and timely insights from multiple sources without having to embark on a major data project with extensive ETL and data storage." Existing such platforms, usually implement a relational model. A DVM provides a virtual layer where the data engineer can easily map data and processes related to an entity. In this respect, it can be considered as a data virtualization platform.

3.6 Model-specific Database Instantiations

A data virtual machine is a conceptual model. While in a traditional database design the data model is predefined and determines storage models, in a conceptual design one can create database instances in different data models (e.g. relational, semi-structured, multi-dimensional, etc.) – and possibly use this model's query language to run queries on top of the instance. For example, one can define a collection of JSON documents rooted on `CustID` for an application (containing customer's transactions within the document), but another user can define a collection of JSON documents rooted on `TransID`. Recall the research question posed in Section 3.4, regarding the delivering format of an individual's data under GDPR compliance. Using a DVM's approach, the service owner can instantiate a database containing the individual's data in the preferred data model of the user.

4 CONCLUSIONS

We introduce a graph-based model to depict data and data processing tasks of the data infrastructure of an organization at a conceptual layer. We argue that this abstraction is useful in a plethora of analytics tasks performed by analysts and data engineers alike. We are currently developing the operators of the algebra over key-list structures in Python. Dataframe queries are translated to an algebraic expression and a simple (unoptimized) plan is generated. The system that handles key-list structures is Redis. Neo4j is used for DVMs. We are developing a tool called DataMingler that allows the management of data sources and the respective DVM, and query formulation in a visual manner.

REFERENCES

[1] Silvana Castano, Valeria De Antonellis, and Sabrina De Capitani di Vimercati. 2001. Global Viewing of Heterogeneous Data Sources. *IEEE Trans. Knowl. Data Eng.* 13, 2 (2001), 277–297. https://doi.org/10.1109/69.917566

[2] Damianos Chatziantoniou. 1999. The PanQ Tool and EMF SQL for Complex Data Management. In *Proceedings of ACM SIGKDD, 1999.* 420–424.

[3] Damianos Chatziantoniou, Michael Akinde, Ted Johnson, and Samuel Kim. 2001. The MD-Join: An Operator for Complex OLAP. In *IEEE International Conference on Data Engineering*. 524–533.

[4] Damianos Chatziantoniou and Florents Tselai. 2014. Introducing Data Connectivity in a Big Data Web. In *Proceedings of the Third Workshop on Data analytics in the Cloud, DanaC 2014.* 7:1–7:4. https://doi.org/10.1145/2627770.2627773

[5] Denodo. 2019. Data Virtualization: The Modern Data Integration Solution. In *White Paper.*

[6] AnHai Doan, Alon Y. Halevy, and Zachary G. Ives. 2012. *Principles of Data Integration.* Morgan Kaufmann. http://research.cs.wisc.edu/dibook/

[7] Jennie Duggan, Aaron J. Elmore, Michael Stonebraker, Magdalena Balazinska, Bill Howe, Jeremy Kepner, Sam Madden, David Maier, Tim Mattson, and Stanley B. Zdonik. 2015. The BigDAWG Polystore System. *SIGMOD Record* 44, 2 (2015), 11–16. https://doi.org/10.1145/2814710.2814713

[8] Daniel Abadi et. al. 2016. The Beckman report on database research. *Commun. ACM* 59, 2 (2016), 92–99. https://doi.org/10.1145/2845915

[9] Gartner. 2018. Data Virtualization Market Guide. In *White Paper.*

[10] Manos Karpathiotakis, Ioannis Alagiannis, Thomas Heinis, Miguel Branco, and Anastasia Ailamaki. 2015. Just-In-Time Data Virtualization: Lightweight Data Management with ViDa. In *CIDR 2015.*

[11] Juan F. Sequeda, Marcelo Arenas, and Daniel P. Miranker. 2012. On Directly Mapping Relational Databases to RDF and OWL. In *WWW.* 649–658.

[12] Vania Maria P. et al. Vidal. 2013. Incremental Maintenance of RDF Views of Relational Data. In *On the Move to Meaningful Internet Systems.*