Hybrid.poly: An Interactive Large-scale In-memory Analytical Polystore

Maksim Podkorytov, Dylan Soderman, Michael Gubanov
Department of Computer Science
University of Texas at San Antonio
San Antonio, USA
{maksim.podkorytov, dylan.soderman, mikhail.gubanov}@utsa.edu

***Abstract*—Anecdotal evidence suggests that the *variety* of Big data is one of the most challenging problems in Computer Science research today [Stonebraker, 2012], [Ou et al., 2017], [Guo et al., 2016], [Bai et al., 2016]. First, Big data comes at us from a myriad of data sources, hence its shape and flavor differ. Second, hundreds of data management systems which work with Big data support different APIs and storage/indexing schemes while exposing data to the users through the *data model* lens, specific to each system. These differences can impede work for users who simply want an accessible interface which can handle relevant unstructured data that is stored within a back-end system. Naturally, such discrepancies in formats, sizes, and shapes can also complicate the development of analytical algorithms which could be implemented on top of large-scale, heterogeneous datasets.**

[Gubanov, 2017b] introduced a consolidated polystore engine, designed to seamlessly ingest and query any type of large-scale data. In this paper we describe a variety of complex analytical workloads that can be processed by such polystore as well as associated research challenges.

***Index Terms*—Large-scale data management, Big data analytics, In-memory data management, Polystore, Polyfuse, Query optimization, Linear algebra, Relational algebra, Web search, Data Integration, Music search, Machine Learning, Locality-sensitive Hashing, Computer vision.**

I. Introduction

Proliferation of different Big data engines and data models designed to handle *Variety* of Big data [Stonebraker, 2012] introduce a significant impediment on the way to *transparency* and ease of access to it. Because of that such data is sometimes called *Dark Data* [Priya et al., 2017], [Gubanov et al., 2017] to reflect difficulties of getting insight into it. Of course, not only accessing, but also supporting any simple or complex analytical algorithms on top of *Dark Data* is also problematic.

Here we envision HYBRID.POLY – an analytical *Polystore* system [She et al., 2016] that would represent a *common ground* for storing, accessing, and analyzing heterogeneous large-scale datasets. It would treat data holistically, thus alleviating difficulties arising in context of pushing data to different data management engines [She et al., 2016]. At the same time, it would support most popular data models, so *any* data can be ingested by design.

This paper is structured as follows. In Section II we describe HYBRID.POLY architecture. Section IV goes over the data models supported by HYBRID.POLY. In section V, we envision the challenges on the way to building and maintaining such system. Section VI is devoted to popular applications supported by HYBRID.POLY. We detail related work in Section III.

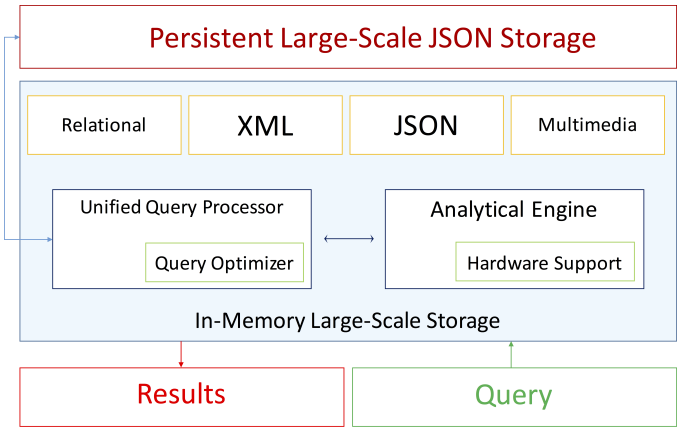

Figure 1. HYBRID.POLY Architecture

II. Architecture

The HYBRID.POLY architecture is illustrated in Figure 1. The *in-memory* storage engine supports a variety of data models. The query interface accepts user queries in a hybrid language that is a superset of SQL providing extra capabilities to make complex analytical queries (e.g. training a Machine Learning classifier) on non-relational data (e.g. JSON, XML, media files) along with relational data [Codd, 1970]. The query is automatically optimized by a hybrid optimizer that is able to process large hybrid (i.e. having not only relational nodes) query plans having thousands of nodes.

A. Query Language

One of the approaches is to have a scripting language (like the one provided by Apache Spark, Pig) and expose HYBRID.POLY engine's API to this script so that the users could run interactive queries. The advantage of this approach is that the logic of querying is abstracted from the logic of storing the data, so that it would be possible to add more data models later and expose querying APIs for these models. The disadvantage is that it would be another Domain Specific

Language (DSL), so some users may be reluctant to use it compared to SQL.

Another approach is to design a query language on top of SQL and augment with pure (not affecting the stored data) query constructs from different data models. We add more data types that exist in standard SQL to allow interoperability between different data models as well.

B. Query Processing Engine

The query processing engine consists of the query parser, the query compiler, and the query optimizer. The query parser processes the user's query and compiles an Abstract Syntax Tree (AST). The query compiler compiles the AST into an internal query representation that is run against the storage. It is possible to reuse the query compilers for the existing query languages and put them into different modules that provide some standard interface to deal with an abstract query compiler; it is also necessary to have a coordinator to manage these components and create the data structure that corresponds to the compiled query in the hybrid query language.

Query Optimizer: We envision two types of query optimizers. The static query optimizer rewrites the AST to get a cheaper execution plan. For example, reordering two operators in a tree, might yield the same result, but improved performance. The dynamic query optimizer uses the information about current data (such as the matrix dimensions) at runtime to pick the optimal execution plan.

The static query optimizer may use the following kinds of optimization. The relational optimization reuses the optimization strategies built into relational engines starting from System R optimizer [Kacimi and Neumann, 2009]. The hybrid one-node cost-based optimization routines examine each node of the logical query plan and pick the physical operator that uses the least resources (time, memory, bandwidth) for the particular logical task. The hybrid cross-node cost-based optimization routines examine groups of logical operators and perform logical query plan rewrites before mapping the logical query plan to the physical operators. Operator fusion, the optimization strategy to map a series of logical operators into one physical operator falls into that category, however there is no prior work that uses this strategy for hybrid query plans.

C. Storage Engine

We envision HYBRID.POLY to employ a distributed in-memory multi-model storage engine with optional persistence. The *distributed* storage allows to handle larger volumes of data. *In-memory* storage decreases latency, enables interactive performance for complex analytical queries. *Persistence* is necessary for fault tolerance. Finally, support of different models is necessary to store heterogeneous data.

D. Hardware acceleration for analytical Query Processing

To speedup analytic query processing, we envision HYBRID.POLY to support an array of recent NVidia GPUs or a specialized matrix processing unit similar to Google TPU [Jouppi et al., 2017]. For example, NVidia Volta, one of such customized matrix processing architectures is going to become available for the general public in the beginning of 2018 [www, 2017c].

III. Related work

In BigDAWG [Duggan et al., 2015] [Gadepally et al., 2016] [She et al., 2016] [Mattson et al., 2017] it is possible to query the data that is distributed across different database engines with different underlying data models; the queries are written in a language supporting nested queries and subqueries written in the languages of BigDAWG *federated* components. The data may be moved between subqueries logically (and federated components physically) using a CAST-expression, provided the moved object semantically exists in both subqueries' data models. The query is optimized within each subquery by reusing the query optimizers of federated components. They also suggest a rule-based mechanism that is able to translate simple components (data types and operations) between the boundaries of different data models, thus influencing the rewriting process within a single component by extending the search space of possible plans. However, as the cardinality of the set of possible rewrites is too high, the burden of creating *interesting* rules is placed on the users of the system.

Our envisioned system is different, because we selected *consolidation* rather than *federation* of storage engines very early in our architecture, which makes all design challenges, storage, query processing and optimization different from federated architecture that BigDAWG follows. We treat data coming from different sources and conforming to different data models holistically, instead of federating multiple data management engines. Hence communication and maintenance costs associated with our system are much less compared to federation of data engines with a mediator. HYBRID.POLY is also focused not only on supporting different data models like BigDAWG, but also on a variety of high-performance complex analytical workloads, hence the language has many built-in analytical constructs, as well as the query optimizer that has the corresponding nodes in the query graph. We also believe that the optimizer should be fully automatic, as a manual approach does not scale with the number of sources and models.

Recently, there have been several attempt on top of a traditional data management engines. One of the recent attempts to add analytics on top of the relational model was made by [Shangyu et al., 2017]. They have built an engine on top of Map/Reduce framework that has relational tables and Linear Algebra objects as first-class citizens. However, they do not consider the problem of supporting a variety of different kinds of data and data models. Also, because the system is built on top of Map/Reduce and all queries are being translated into Map/Reduce jobs, it is not suitable for interactive workloads.

Another venue of the related work is associated with enabling the users of data managements systems to perform Machine Learning tasks in a declarative way. The notable approach was made in [Ghoting et al., 2011], where the

authors have considered the introduction of Declarative Machine Learning on MapReduce. Typically data scientists had to write the Machine Learning solutions in Java and such code was data agnostic, i.e. it depended heavily on data sources and data distribution between MapReduce nodes. In this paper Ghoting et al. have invented a scripting language that allows to operate scalar values and matrices in a way that is independent of their physical representation. All the decisions about physical operator selection are deferred to the SystemML query optimizer.

Kunft et al. in [Kunft et al., 2016] also explore the problem of unification of relational and linear algebras. They are trying to start from the point of view of how to optimize the hybrid workflows that operate on both relations and matrices. Their solution is to create generic types for matrices and relations and apply category theory to optimize the queries that operate on both data types. They also include reasoning about different physical representations of matrices as well as relations in their framework.

IV. Supported data models

Relational data model. One of the popular data types we envision to support is tabular data, i.e. data organized in tables in such a way that all values in a column have the same data type (e.g., a number, a string, a datetime). Relational data model [Codd, 1970] has proven itself as it has been used for decades in commercial RDBMSes , but it has its drawbacks. One of them is difficulties in expressing numerical algorithms in terms of SQL, the de-facto standard language used in RDBMSes [Astrahan et al., 1976b].

Array data model. Another data type we envision to support is *arrays*. The arrays are rectangular collections of atomic values, namely, numbers (e.g. floating point, integer, precise decimals), booleans and strings. Each atomic value within an array is associated with a unique tuple of integers that is the multidimensional index of that value within the array, all such tuples have the same length for a given array and the numeric value of that length is the number of dimensions of that array; each integer within a tuple is indexing a dimension of the array, integers within a tuple have the lower bound of 0 and the upper bound of the length of the dimension corresponding to that index. Some common examples of arrays are matrices (2-dimensional arrays), vectors (1-dimensional arrays) and scalars (0-dimensional arrays). Arrays with more than 2 dimensions are also used for scientific purposes, for instance, for training machine learning classifiers such as Neural Networks, for Tensor calculations in Chemistry and Physics. Below we list some of the fundamental operations on arrays we envision to support:

1) element-wise summation, e.g.

$$\begin{pmatrix}1 & 2\\3 & 4\end{pmatrix} + \begin{pmatrix}4 & 3\\2 & 1\end{pmatrix} = \begin{pmatrix}5 & 5\\5 & 5\end{pmatrix} \tag{1}$$

2) element-wise subtraction, e.g.

$$\begin{pmatrix}1 & 2\\3 & 4\end{pmatrix} - \begin{pmatrix}4 & 3\\2 & 1\end{pmatrix} = \begin{pmatrix}-3 & -1\\1 & 3\end{pmatrix} \tag{2}$$

3) element-wise product (aka Hadamard product), e.g.

$$\begin{pmatrix}1 & 2\\3 & 4\end{pmatrix} \otimes \begin{pmatrix}4 & 3\\2 & 1\end{pmatrix} = \begin{pmatrix}4 & 6\\6 & 4\end{pmatrix} \tag{3}$$

4) element-wise inverse ($x \to \frac{1}{x}$), e.g.

$$inv\left(\begin{pmatrix}1.0 & 2.0\\0.5 & 4.0\end{pmatrix}\right) = \begin{pmatrix}1.0 & 0.5\\2.0 & 0.125\end{pmatrix} \tag{4}$$

5) element-wise application of standard (found in mathematical libraries of many programming language) functions: sin, cos, exp, ln, sqr, sqrt, e.g.

$$sqrt\left(\begin{pmatrix}1.0 & 4.0\\0.25 & 6.25\end{pmatrix}\right) = \begin{pmatrix}1.0 & 2.0\\0.5 & 2.5\end{pmatrix} \tag{5}$$

6) element-wise conversion between different data types of matrix cells (real and complex floating point numbers, integer numbers, booleans and strings), e.g.

$$int\left(\begin{pmatrix}1.0 & 4.0\\0.25 & 6.25\end{pmatrix}\right) = \begin{pmatrix}1 & 2\\0 & 6\end{pmatrix} \tag{6}$$

7) statistical functions: size, min, max, argmin, argmax, mean, std, e.g.

$$mean\left(\begin{pmatrix}1.0 & 4.0\\0.25 & 6.25\end{pmatrix}\right) = \frac{1+4+0.25+6.25}{4} = 2.875 \tag{7}$$

8) transposition (flipping dimensions), e.g.

$$transpose\left(\begin{pmatrix}1.0 & 4.0\\0.25 & 6.25\end{pmatrix}\right) = \begin{pmatrix}1.0 & 0.25\\4.0 & 6.25\end{pmatrix} \tag{8}$$

9) slicing (getting a rectangular subregion of a matrix), e.g. getting elements with positions from 1 to 3 in a second row of a matrix. e.g.

$$slice\left(\begin{pmatrix}1.0 & 4.0 & 9.0\\0.25 & 6.25 & 0.0\end{pmatrix}, 2, 1:3\right) = \begin{pmatrix}0.25 & 6.25 & 0.0\end{pmatrix} \tag{9}$$

10) cross-product (aka matrix-matrix multiplication), e.g.

$$\begin{pmatrix}1.0 & 4.0 & 9.0\\0.25 & 6.25 & 0.0\end{pmatrix} \times \begin{pmatrix}2.0\\1.0\\0.0\end{pmatrix} = \begin{pmatrix}6.0\\6.75\end{pmatrix} \tag{10}$$

11) factorization [www, 2017b] [wik, 2017b], e.g. SVD-factorization:

$$\begin{aligned} SVD\_factorize\left(\begin{pmatrix}0 & 0 & -2\\-4 & 0 & 0\end{pmatrix}\right) = \\ = \begin{pmatrix}0 & 1\\1 & 0\end{pmatrix} \times \begin{pmatrix}4 & 0 & 0\\0 & 2 & 0\end{pmatrix} \times \begin{pmatrix}-1 & 0 & 0\\0 & 0 & -1\\0 & 1 & 0\end{pmatrix} \end{aligned} \tag{11}$$

12) splitting [wik, 2017c], e.g. LDU-splitting:

$$\begin{aligned} LDU\_split\left(\begin{pmatrix}1 & 2 & 3\\4 & 5 & 6\\7 & 8 & 9\end{pmatrix}\right) = \\ = \begin{pmatrix}0 & 0 & 0\\4 & 0 & 0\\7 & 8 & 0\end{pmatrix} + \begin{pmatrix}1 & 0 & 0\\0 & 5 & 0\\0 & 0 & 9\end{pmatrix} + \begin{pmatrix}0 & 2 & 3\\0 & 0 & 6\\0 & 0 & 0\end{pmatrix} \end{aligned} \tag{12}$$

13) computable properties of square matrices (trace, determinant, eigenvalues and eigenvectors), e.g.

$$trace\left(\begin{pmatrix}1 & 2 & 3\\ 4 & 5 & 6\\ 7 & 8 & 9\end{pmatrix}\right) = 1+5+9 = 15 \qquad (13)$$

There are several considerations regarding the data model. First, for element-wise operations, there exist their counterparts where one of the arguments is scalar, so it makes sense to implement these scalar-on-matrix ops consistently with element-wise ops. Second, some higher-level operations such as numerous matrix factorizations (LU, QR, Cholesky, SVD, NMF) as well as Fourier transformations are widely used in numerical applications [Sra and Dhillon, 2006] [Alter et al., 2000] [Trefethen and Bau, 1997]. Finally, the iterative methods of solving systems of linear equations (such as Gauss-Seidel [Young and Gregory, 1988]) use matrix splitting, so matrix splitting routines (e.g. $A \rightarrow L + D + U$, where L is the lower triangle, D - diagonal and U - upper triangle) may be useful for running iterative solvers on some matrices stored in our database.

XML data model. XML data, unlike relational is designed to include not only data, but also schema of that data in the same file. It is also possible to define complex nested data structures using XML. An example of XML file can be seen in Listing 1. Simultaneous human and machine readability led to usage of XML format for systems configuration (e.g. Apache Maven), Java EE uses XML to serialize the data that is used by different services. There exists a standard query language to process XML data, called XQuery [Chamberlin et al., 2001], it was revised multiple times after its introduction. The data model organizes items in a tree-like structure, and the query language applies functions to collections of nodes that satisfy certain criteria. The access to elements within the tree is done in a fashion similar to accessing files within a file system.

Listing 1. XML example

```
<!DOCTYPE glossary PUBLIC "-//OASIS//DTD
    DocBook V3.1//EN">
 <glossary><title>example glossary</title>
  <GlossDiv><title>S</title>
   <GlossList>
    <GlossEntry ID="SGML" SortAs="SGML">
     <GlossTerm>Standard Generalized Markup
         Language</GlossTerm>
     <Acronym>SGML</Acronym>
     <Abbrev>ISO 8879:1986</Abbrev>
     <GlossDef>
      <para>A meta-markup language, used to
          create markup
languages such as DocBook.</para>
      <GlossSeeAlso OtherTerm="GML">
      <GlossSeeAlso OtherTerm="XML">
     </GlossDef>
     <GlossSee OtherTerm="markup">
    </GlossEntry>
   </GlossList>
  </GlossDiv>
 </glossary>
```

JSON data model. Another data model that organizes data in a tree-like manner is JSON. JSON uses Javascript syntax to organize atoms (strings and numbers), and arrays of atoms into mappings, also known as dictionaries, having string identifiers as keys. An example of JSON file can be seen in Listing 2 [www, 2017e]. Due to its simplicity and easy interoperability with Javascript the format gained popularity [Mongodb,]. However, there are still considerable differences in query languages among storage engines and there is no de-facto standard. However, de-jure there exists an RFC [www, 2017a] that suggest how to query elements within a JSON document, and it has been implemented in some of the storages. It could be used as a starting point for querying the JSON data stored in our polystore, as it promises to be the common feature of JSON stores.

Listing 2. JSON example

```
{
 "glossary": {
  "title": "example glossary",
  "GlossDiv": {
   "title": "S",
   "GlossList": {
    "GlossEntry": {
     "ID": "SGML",
        "SortAs": "SGML",
        "GlossTerm": "Standard Generalized
            Markup Language",
        "Acronym": "SGML",
        "Abbrev": "ISO 8879:1986",
        "GlossDef": {
      "para": "A meta-markup language, used to
          create markup languages such as
          DocBook.",
          "GlossSeeAlso": ["GML", "XML"]
      },
        "GlossSee": "markup"
}}}}}
```

Graph data model. Some data as social network connections and ontologies is better represented as graphs, i.e. nodes, edges and properties assigned to both nodes and edges. There exists a separate class of Graph databases specifically optimized for dealing with large amounts of data organized in graphs, running graph processing routines such as depth-first search, breadth-first search, shortest path discovery and maximum flow. Graph databases provide query interfaces, and it is necessary to use them in HYBRID.POLY's query language.

More data types: images, audio files, geo tags. A large amount of data is represented by images, videos and audio files; for instance, images of cells in Biology, images of sky in Astrophysics, media storages such as Spotify, Apple Music, Youtube and Vimeo. There does not exist a data model for storing the media files, although there exist the metadata that comes along with them (e.g. EXIF format for embedding the properties of photography equipment into image files, ID3 tags for decorating music tracks with album cover image and genre, author and title information as well as technical details like encoding, quality, discretization frequency and resolution).

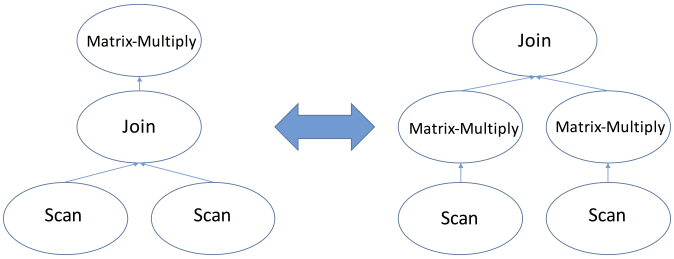

Figure 2. `Hybrid` cross-node optimization

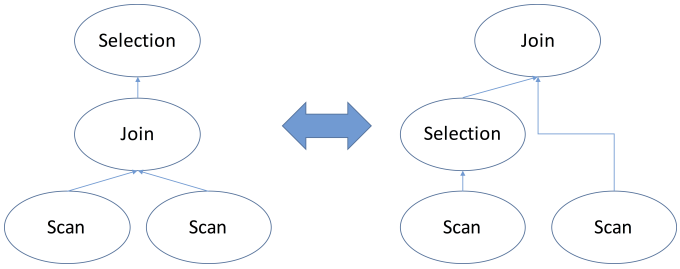

Figure 3. `Relational` cross-node optimization

V. Challenges

Unified Data Model. There is an abundance of different data models and it is necessary to think of the data model that incorporates all the different data types and the relations between them. In modern databases, there exists a somewhat limited approach for achieving that goal; for instance, the MS SQL Server allows to store XML [www, 2017f] and text files in the database. However, the existing approaches do not handle all of the data types and there is no query language that allows to query all of them simultaneously. UFO project at IBM Almaden [Gubanov et al., 2009], [Gubanov and Shapiro, 2012] is one of the attempts to design and evaluate a unified data storage abstraction spanning different data models. Model management is another project aimed at designing operators manipulating schemas and mappings between them [Gubanov et al., 2008].

Design of the Storage Engine. The data management systems are designed to store particular data types and are expected to operate under particular workload conditions; thus, they have storages carefully optimized for these data types and workload conditions. As HYBRID.POLY is able to store and query different kinds of data, the design of the component responsible for storage of that data is a challenge. There are two options for the storage engine of a hybrid system. The first option is to have a federation of storages for each data type managed by a mediator [She et al., 2016]. Another option is to have a single consolidated storage that is able to handle each data type. We envision the latter have better performance due to the reduced communication and query processing cost.

Design of the Query Compiler. Having designed the unified data model, it is necessary to be able to ingest the data in HYBRID.POLY as well as query the data from the system. If data is stored in different storages optimized for specific data types [She et al., 2016], it is possible to reuse the logic that is used for compiling the queries that operate within one data type, though it is necessary to apply effort on wiring together these elementary pieces into the HYBRID.POLY's final query compiler. For this case, it is preferred to have the query compilers for each data type abstracted into modules with a standard interface, and one global coordinator of these modules. Otherwise, it is necessary to develop the query compiler taking into account the consolidated storage.

Design of the query optimizer. Another challenge is the design of the query optimizer. Substantial work body has been done on query optimization in the standard RDBMSes [Astrahan et al., 1976a] as well as some modern systems [Michiels, 2003]. However, as HYBRID.POLY supports a larger variety of data types, it is necessary to optimize the queries in the hybrid query language, building on top of the previous work done on optimization. An initial approximation to the solution to this problem could be performing the optimization dynamically, collecting the statistics, and then combining these results in the global query optimizer. In this approach, the storage engine must expose these statistics to the optimizer, otherwise statistics collection is not feasible. It is also challenging to consolidate the statistics gathered for different storage types *if* the federation storage architecture is chosen, because storages for different data types expose different kinds of statistics. Again, there is the similar pattern where the solutions to subproblems are abstracted into modules with a standard interface, and one global coordinator uses this interface to do the desired task.

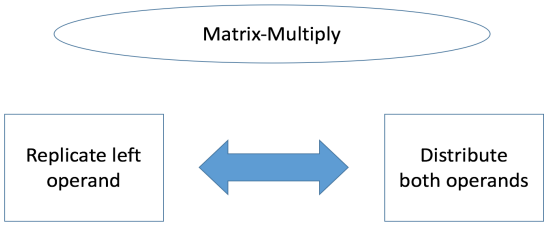

Figure 4. `Hybrid` single-node optimization

In HYBRID.POLY, we envision three kinds of optimizations – relational cross-node, hybrid cross-node, and hybrid single-node optimization. Figure 3 illustrates an example of purely relational optimization (pushing down a selection), Figure 4 illustrates an example of single-node hybrid optimization (distributed matrix multiplication, substitution of left operand replication by distribution of both operands), Figure 2 illustrates an example of hybrid cross-node optimization (pushing down matrix multiplication). There are many other possible optimizations of either of these three types, we picked these as simple examples. Section II-B describes possible optimizations in more detail. This optimizer must also scale to large plans having thousands of nodes.

With respect to Linear Algebra support, a few challenges

also appear. First, there exist different formats of storing matrices, as some of matrix data is dense (e.g. numeric representations of grayscale images where each pixel is associated with a number that carries information about its intensity) and some is sparse (e.g. matrices that arise in finite difference methods (FDM) during the numerical solution of partial differential equations (PDE) [Saad, 2003]). Therefore it is necessary to be able to store all kinds of matrix data and to perform operations on data stored in possibly different forms. Looking from another perspective, as the number of operands in an expression that operates on matrices grows, the number of possible combinations of the storage types for the operands grows exponentially [Elgamal et al.,], and so does the number of physical plans for the query, handling this big number of possible plans is a challenge.

VI. Applications

A. Web-search

One of the possible applications of HYBRID.POLY is its use as a back-end for a web search engine. A web crawler uses fast data ingest and fast analytical capabilities to add documents to the database with computed text properties such as TF/IDF [www, 2017d], where a user can query for documents stored within the database. HYBRID.POLY is also able to perform ranking of search results using Convolutional Neural Networks (CNN) or other ranking algorithms.

1) Ranking of search results: The following query computes a text document search rank using an implemented ranking function called **classify**:

```
SELECT classify(d.v) AS score, d.id FROM
    Documents d
WHERE d.id = 13243;
```

Running Large-Scale Machine Learning tasks from a client interface to the database becomes feasible, as users have the means to define these tasks using the HYBRID.POLY query language. These particular tasks run in-database, which avoids costly Input/Output to and from an external computing engine (such as Spark or Dryad).

2) Machine Learning: For example, a group of text documents can be classified by their topic using the Naive Bayes classification routine. The input for this procedure is the data store called Document which contains precomputed word counts and an associated group id. The classification is performed as follows:

Histogram Construction. At this step the vector of cumulative word counts is constructed for each document class. In addition, the number of documents that fall into each class is computed, where the result is then placed into a new set histogram. The query is as follows:

```
SELECT SUM(Document.vector) AS word_count,
    COUNT(Document.vector) AS size, Document.
    label
FROM Document
INTO Histogram
GROUP BY Document.label;
```

Histogram Smoothing. Next, the Naive Bayes model is trained. The model is another set of objects that may be used to predict a document's class given the vector of word counts for that document. The model is obtained as follows: considering the set obtained at the previous step, for every item we apply the unary function $f(x) = \ln \frac{x+a}{size+l*a}$ where $size$ is the second element of the tuple, l is the vector's dimensionality and a is the parameter of Lidstone (Laplace) smoothing [Chen and Goodman, 1996], defined by the user. The modified vector and answer are then placed into a new set model.

```
SELECT LOG((word_count + 1) / (size + LEN(
    word_count))) AS weight, label
FROM Histogram
INTO Model;
```

Likelihoods Construction. Next, the classification of documents with unknown document class is performed. This is achieved by pre-computing and storing the set of likelihoods that a document belongs to a document class for each document and each document class. For each document of the test set and each document class stored in the Histogram set, we compute the measure of cosine similarity between the document and the document class. For the computation of cosine similarity we use dot product of two vectors. This is also the main part of Naive Bayes prediction task. The result is put into the new set Likelihood.

```
SELECT DOT(Input.word_count, Model.weight) AS
    likelihood, Model.label, Input.id
FROM Input, Model
INTO Likelihoods
ORDER BY Input.id;
```

Final prediction. Finally, now that every combination of every document and document class in the set Likelihood has the likelihood estimation of the document belonging to the class, it is wise to choose the class with maximum likelihood is chosen for a given document.

```
SELECT MAX(likelihood), id, label
FROM Likelihoods
GROUP BY id;
```

B. Clustering using Locality-Sensitive Hashing (LSH)

One of the popular scalable clustering algorithms is Locality-Sensitive Hashing [Slaney and Casey, 2008] (LSH). It has been applied to several problem domains, such as audio similarity identification, image similarity identification, video fingerprinting, audio fingerprinting, gene expression similarity identification and other kinds of similarity identification. Our query language is capable of expressing the LSH workflow, so the users can perform all of the mentioned similarity identification tasks on large datasets of heterogeneous data.

1) LSH query example: In this example it is assumed that the set Hashes contains the pre-generated family of hash functions that conforms to certain rules that enable the use of this family in LSH clustering and querying process. The set Documents contains the vector representation of text documents (e.g., Bag-of-Words) on which we perform

clustering. The set UnclassifiedDocuments contains the vector representation of text documents for which we want to know their closest neighbors from the set of Documents.

Clustering. On this step we apply all hash functions from the set Hashes to each document from the set of documents that we want to cluster. Each hash function application is computing a dot product between the hash vector and the vector representation of a document. The hash-document pairs that the hash function application deemed positive are then stored in the set Model.

```
SELECT h.id AS hid, d.id AS did, SIGN(DOT(h.v,
    d.v)) AS dp
FROM Hashes h, Documents d
INTO Model
WHERE dp = 1
ORDER BY hid, did;
```

Querying the Nearest Neighbors. Given a vector representation of a document, the query finds the documents which yield a high correlation with this document (a threshold of at least 0.95) using the information obtained on the previous step. The query is more efficient than the naive linear search, because instead of going over the set of all documents, its search space includes only the documents that are proven similar to the incoming document by the set of hash functions stored inside the set Hashes.

```
SELECT m.id FROM Model m,Hashes h,
    UnclassifiedDocuments ud,Documents d
WHERE ud.id = 13243
AND DOT(ud.v, h.v) = 1
AND m.hid = h.id
AND m.did = d.id
AND DOT(ud.v, d.v) > 0.95;
```

C. Music search

In recent years, we have seen an increasing number of musical services that deal with millions of music files [Bertin-Mahieux et al., 2011]; such services include Spotify, Apple Music, Google Music, Yandex Music, Amazon Prime Music, Shazam, etc. One of the common interfaces to such a service is searching for a particular music track based on features such as author, album or title. This metadata, however, is not always available, so we want more possible queriesfor finding a music track in such a database.

Two new possibilities come to our mind: the first one is looking up the song by its lyrics, which can be done by applying the same math as we did with web search, then combining the music data and the lyrics data using our query language, as both music and text can be stored in our polystore. To summarize, music search can be done in terms of text search. The second way of performing a music search would be extracting some vector data from music files [www, 2012] and storing that data along with music files in our polystore, making it possible to run Large-Scale clustering on that data and find similar music tracks [Camastra and Vinciarelli,]. Locality-sensitive hashing can be used to carry out the task [Casey and Slaney, 2007] [Casey et al., 2008] [Yu et al., 2009], along with an abundance of other methods [Camastra and Vinciarelli,] [Serrà et al., 2010].

When searching music that is stored within a table based on vector data, a new clause can be added onto the Select query called *MelodyMatch* to retrieve said data. This proximity-match takes in two arguments: the second is used to match against the possible melodies contained within the first through the use of melody extraction. MelodyMatch will evaluate into a decimal value within the decimal range 0-1 called the *voicing detection* : an estimation on whether a melody is present within the music file. Once a certain threshold has been reached for the condition (in this case, 0.95), information such as the artist name, song title, and the ID of the music file can be returned back from the query.

```
SELECT Artists, SongTitle, melodyID
FROM wavFiles AS w, singleMelodies AS sm
WHERE MELODYMATCH(w.waves, sm.vector) > 0.95;
```

Another application of our system to perform a music search is to extract a group of patterns from a music file (e.g. using some dimensionality reduction technique such as Principal Component Analysis [Jolliffe, 2002] or clustering techniques such as Locality-Sensitive Hashing [Slaney and Casey, 2008]) and using these patterns to detect similarities between melodies [Casey and Slaney, 2007] for the purpose of copyright infringement. One step further would be abstracting such tasks into source-oblivious operators in our query language.

D. Image and video processing

There are services that accumulate terabytes of images such as Flickr, Google Photos, Instagram, and Facebook, as well as public scientific datasets with large amounts of images [wik, 2017a]. Performing image processing tasks and simple image searches on these big datasets would prove useful, since images often contain additional metadata, where text searches can become applicable which in turn simplifies searches on images. Storing metadata along with images is enabled by our architecture. Moreover, some Large-Scale clustering and Machine Learning can be performed on these large datasets. Images can be represented as 2-dimensional arrays, and some image processing operations rely heavily on Linear Algebra [Gonzalez and Wintz, 1977], however, non-linear routines may also be used for image processing. Our hybrid query language has native support for image processing routines, thus making it possible to process myriads of images at scale [Gubanov, 2017a].

There also exists services which accumulate terabytes of videos such as Youtube, as well as public scientific datasets with large amount of videos [wik, 2017a]. Footprinting, classification, and clustering tasks may be enabled by our universal query language and universal storage.

VII. Acknowledgments

We would like to thank anonymous reviewers for their comments on the draft of this paper.

REFERENCES

[www, 2012] (2012). online: A very short introduction to sound analysis.

[www, 2017a] (2017a). online: Json pointer rfc.

[www, 2017b] (2017b). online: Matrix factorizations jungle.

[www, 2017c] (2017c). online: Nvidia volta ai archtecture.

[www, 2017d] (2017d). online: Tfxidf repository.

[www, 2017e] (2017e). online: xml and json file examples.

[www, 2017f] (2017f). online: Xml data.

[wik, 2017a] (2017a). Wikipedia: list of image datasets for machine learning research.

[wik, 2017b] (2017b). Wikipedia: Matrix decomposition.

[wik, 2017c] (2017c). Wikipedia: Matrix splitting.

[Alter et al., 2000] Alter, O., Brown, P. O., and Botstein, D. (2000). Singular value decomposition for genome-wide expression data processing and modeling. *Proceedings of the National Academy of Sciences*, 97(18):10101–10106.

[Astrahan et al., 1976a] Astrahan, M. M., Blasgen, M. W., Chamberlin, D. D., Eswaran, K. P., Gray, J., Griffiths, P. P., King, W. F., Lorie, R. A., McJones, P. R., Mehl, J. W., Putzolu, G. R., Traiger, I. L., Wade, B. W., and Watson, V. (1976a). System r: Relational approach to database management. *ACM Trans. Database Syst.*, 1:97–137.

[Astrahan et al., 1976b] 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., Putzolu, G. R., Traiger, I. L., Wade, B. W., and Watson, V. (1976b). System r: Relational approach to database management. *ACM Trans. Database Syst.*, 1:97–137.

[Bai et al., 2016] Bai, L., Cheng, X., Liang, J., and Shen, H. (2016). An optimization model for clustering categorical data streams with drifting concepts. *IEEE TKDE*, 28(11):2871–2883.

[Bertin-Mahieux et al., 2011] Bertin-Mahieux, T., Ellis, D. P. W., Whitman, B., and Lamere, P. (2011). The million song dataset. In *ISMIR*.

[Camastra and Vinciarelli,] Camastra, F. and Vinciarelli, A. Machine learning for audio, image and video analysis.

[Casey et al., 2008] Casey, M. A., Rhodes, C., and Slaney, M. (2008). Analysis of minimum distances in high-dimensional musical spaces. *IEEE Transactions on Audio, Speech, and Language Processing*, 16:1015–1028.

[Casey and Slaney, 2007] Casey, M. A. and Slaney, M. (2007). Fast recognition of remixed music audio. *2007 IEEE International Conference on Acoustics, Speech and Signal Processing - ICASSP '07*, 4:IV–1425–IV–1428.

[Chamberlin et al., 2001] Chamberlin, D., Clark, J., Florescu, D., Com, J., and Robie (2001). Xquery 1.0: An xml query language.

[Chen and Goodman, 1996] Chen, S. F. and Goodman, J. (1996). An empirical study of smoothing techniques for language modeling. In *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, pages 310–318. Association for Computational Linguistics.

[Codd, 1970] Codd, E. F. (1970). A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387.

[Duggan et al., 2015] Duggan, J., Elmore, A. J., Stonebraker, M., Balazinska, M., Howe, B., Kepner, J., Madden, S., Maier, D., Mattson, T. G., and Zdonik, S. B. (2015). The bigdawg polystore system. *SIGMOD Record*, 44:11–16.

[Elgamal et al.,] Elgamal, T., Luo, S., Boehm, M., Evfimievski, A. V., Tatikonda, S., Reinwald, B., and Sen, P. Spoof: Sum-product optimization and operator fusion for large-scale machine learning.

[Gadepally et al., 2016] Gadepally, V., Chen, P., Duggan, J., Elmore, A. J., Haynes, B., Kepner, J., Madden, S., Mattson, T. G., and Stonebraker, M. (2016). The bigdawg polystore system and architecture. *HPEC*.

[Ghoting et al., 2011] Ghoting, A., Krishnamurthy, R., Pednault, E., Reinwald, B., Sindhwani, V., Tatikonda, S., Tian, Y., and Vaithyanathan, S. (2011). Systemml: Declarative machine learning on mapreduce. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pages 231–242. IEEE.

[Gonzalez and Wintz, 1977] Gonzalez, R. and Wintz, P. (1977). Digital image processing.

[Gubanov, 2017a] Gubanov, M. (2017a). Hybrid: A large-scale in memory image analytics system. In *CIDR*.

[Gubanov, 2017b] Gubanov, M. (2017b). Polyfuse: A large-scale hybrid data fusion system. In *ICDE DESWeb*.

[Gubanov et al., 2008] Gubanov, M., Bernstein, P. A., and Moshchuk, A. (2008). Model management engine for data integration with reverse-engineering support. In *ICDE*.

[Gubanov et al., 2017] Gubanov, M., Priya, M., and Podkorytov, M. (2017). Cognitivedb: An intelligent navigator for large-scale dark structured data. In *WWW*.

[Gubanov and Shapiro, 2012] Gubanov, M. and Shapiro, L. (2012). Using unified famous objects (ufo) to automate alzheimer's disease diagnostics. In *BIBM*.

[Gubanov et al., 2009] Gubanov, M. N., Popa, L., Ho, H., Pirahesh, H., Chang, J.-Y., and Chen, S.-C. (2009). Ibm ufo repository: Object-oriented data integration. *VLDB*.

[Guo et al., 2016] Guo, J., Fan, Y., Ai, Q., and Croft, W. B. (2016). Semantic matching by non-linear word transportation for information retrieval. In *CIKM*.

[Jolliffe, 2002] Jolliffe, I. T. (2002). Principal component analysis and factor analysis. *Principal component analysis*, pages 150–166.

[Jouppi et al., 2017] Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., Borchers, A., Boyle, R., luc Cantin, P., Chao, C., Clark, C., Coriell, J., Daley, M., Dau, M., Dean, J., Gelb, B., Ghaemmaghami, T. V., Gottipati, R., Gulland, W., Hagmann, R., Ho, C. R., Hogberg, D., Hu, J., Hundt, R., Hurt, D., Ibarz, J., Jaffey, A., Jaworski, A., Kaplan, A., Khaitan, H., Koch, A., Kumar, N., Lacy, S., Laudon, J., Law, J., Le, D., Leary, C., Liu, Z., Lucke, K., Lundin, A., MacKean, G., Maggiore, A., Mahony, M., Miller, K., Nagarajan, R., Narayanaswami, R., Ni, R., Nix, K., Norrie, T., Omernick, M., Penukonda, N., Phelps, A., Ross, J., Ross, M., Salek, A., Samadiani, E., Severn, C., Sizikov, G., Snelham, M., Souter, J., Steinberg, D., Swing, A., Tan, M., Thorson, G., Tian, B., Toma, H., Tuttle, E., Vasudevan, V., Walter, R., Wang, W., Wilcox, E., and Yoon, D. H. (2017). In-datacenter performance analysis of a tensor processing unit.

[Kacimi and Neumann, 2009] Kacimi, M. and Neumann, T. (2009). *System R (R*) Optimizer*, pages 2900–2905. Springer US, Boston, MA.

[Kunft et al., 2016] Kunft, A., Alexandrov, A., Katsifodimos, A., and Markl, V. (2016). Bridging the gap: Towards optimization across linear and relational algebra. In *BeyondMR*.

[Mattson et al., 2017] Mattson, T. G., Gadepally, V., She, Z., Dziedzic, A., and Parkhurst, J. (2017). Demonstrating the bigdawg polystore system for ocean metagenomics analysis. In *CIDR*.

[Michiels, 2003] Michiels, P. (2003). Xquery optimization. In *VLDB PhD Workshop*.

[Mongodb,] Mongodb. Mongodb architecture guide.

[Ou et al., 2017] Ou, C., Jin, X., Wang, Y., and Cheng, X. (2017). Modeling heterogeneous information spreading abilities of social network ties. *Simulation Modeling Practice and Theory*, 75:67 – 76.

[Priya et al., 2017] Priya, M., Podkorytov, M., and Gubanov, M. (2017). ilight: A flashlight for large-scale dark structured data. In *MIT Annual DB Conference*.

[Saad, 2003] Saad, Y. (2003). *Iterative methods for sparse linear systems*. SIAM.

[Serrà et al., 2010] Serrà, J., Gómez, E., and Herrera, P. (2010). Audio cover song identification and similarity: Background, approaches, evaluation, and beyond. In *Advances in Music Information Retrieval*.

[Shangyu et al., 2017] Shangyu, L., Zekai, G., Michael, G., Luis, P., and Christopher, J. (2017). Scalable linear algebra on a relational database system. In *ICDE*.

[She et al., 2016] She, Z., Ravishankar, S., and Duggan, J. (2016). Bigdawg polystore query optimization through semantic equivalences. *HPEC*.

[Slaney and Casey, 2008] Slaney, M. and Casey, M. (2008). Locality-sensitive hashing for finding nearest neighbors [lecture notes]. *IEEE Signal Processing Magazine*, 25(2):128–131.

[Sra and Dhillon, 2006] Sra, S. and Dhillon, I. S. (2006). Generalized non-negative matrix approximations with bregman divergences. In Weiss, Y., Schölkopf, P. B., and Platt, J. C., editors, *Advances in Neural Information Processing Systems 18*, pages 283–290. MIT Press.

[Stonebraker, 2012] Stonebraker, M. (2012). Big data means at least three different things... In *NIST Big Data Workshop*.

[Trefethen and Bau, 1997] Trefethen, L. N. and Bau, D. (1997). *Numerical Linear Algebra*. SIAM.

[Young and Gregory, 1988] Young, D. M. and Gregory, R. T. (1988). *A survey of numerical mathematics*, volume 1. Courier Corporation.

[Yu et al., 2009] Yu, Y., Crucianu, M., Oria, V., and Chen, L. (2009). Local summarization and multi-level lsh for retrieving multi-variant audio tracks. In *ACM Multimedia*.