2019 IEEE 35th International Conference on Data Engineering (ICDE)

Hybrid.Poly: A Consolidated Interactive Analytical Polystore System

Maksim Podkorytov, Michael Gubanov
Department of Computer Science
Florida State University

***Abstract*—Anecdotal evidence suggests the *Variety* of Big data is one of the most challenging problems in Computer Science research today [1]. First, Big data arrives from a myriad of data sources, hence its shape and flavor differ. Second, hundreds of different Big data management systems support different APIs, storage/indexing schemes, and expose data to the users through their data model lens, each specific to their own system. All of these offer a significant impediment for Big data users who just want an easy to use interface to all relevant data regardless of its shape, format, size, and a back-end system used to store it. Naturally, these differences also complicate development of any analytical algorithms on top of large-scale, heterogeneous datasets.**

Here we describe HYBRID.POLY – a consolidated in-memory polystore engine [2], designed to support heterogeneous large-scale data and interactively process complex analytical workloads. We execute and evaluate several popular analytical workloads including Data Fusion, Machine Learning, and Music search at scale.

I. INTRODUCTION

Proliferation of different *data models* and large-scale data management systems [1], [2], [3], [4], [5], [6] complicate access to heterogeneous Big data. To reflect this "non-transparency" it is also sometimes referred to as *Dark Data* [7]. The access and execution of any workloads on such *Dark Data* is problematic.

In this paper we describe HYBRID.POLY – an analytical in-memory *Polystore* [2], [3] engine capable of ingesting and processing complex analytical workloads over diverse large-scale datasets at interactive speed. We describe a hybrid relational analytical query language stemming from SQL which is based on the relational model. It enables advanced *in-database* analytical workloads. We illustrate it by describing several popular analytical workloads implemented in our language – Data Fusion, Machine Learning, and Music similarity search. We note that in our language, the complex analytical workloads are naturally expressed by a concise hybrid SQL query, which is a rare and important property among such "complex analytics" systems. We also emphasize HYBRID.POLY's interactive performance at scale.

Related work: There have been several recent attempts to add analytical extensions (e.g. linear algebra operators) to Map/Reduce-based engines supporting SQL as an add-on (e.g. SimSQL [8]). Our system is a native parallel relational in-memory store, which places it in a different category compared to any Map/Reduce-based derivative systems due to significant architectural differences [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21]. In contrast to the systems having relational or linear algebra implemented on top of Map/Reduce, our engine, being a native parallel in-memory *relational* store supports *interactive* workloads by nature. For example, real-time similarity matching for music would be *non-interactive* using any Map/Reduce derivative due to the relatively high latency stemming from scheduling, distributing, and executing Map/Reduce jobs. Other systems do not support the *non-linear* analytical workloads such as the music similarity matching described here.

The engines supporting machine learning workloads as high-level operators are another venue of related work associated with declarative machine learning. The authors of SystemML implemented this approach for several back-ends – classical Hadoop, Map/Reduce on Spark [22], and explored a few of the optimization venues [23]. Our approach differs by supporting a much wider range of analytics, not just machine learning, and using an interactive parallel in-memory engine as a back-end, instead of Hadoop or Spark. This enables interactive performance of complex analytical workloads. Our hybrid query language is *declarative* and our SQL-based language includes linear (e.g. matrices and vectors) as well as non-linear operators and clauses.

Another line of related research includes *polystores* – systems envisioned to store and query data coming from different data sources. One such systems is BigDAWG [24]. While BigDAWG mediates over a federation of different existing engines, HYBRID.POLY chooses one consolidated parallel engine as a back-end to store and query all data. This yields an easier system to use with significant performance advantages that "talks in one language" instead of "having a user talk in different languages" depending on the number of different engines in the BigDAWG federation. Other key distinctions of our approach are the in-memory storage, interactive query performance, the hybrid language based on SQL tailored to query different data models, as well as complex analytics support. We have introduced our vision of HYBRID.POLY in [2], [3].

Query by Humming (QbH) is a music retrieval approach that involves taking a melody hummed by the user and looking it up in the existing database [25]. One of the scenarios implemented in our hybrid language demonstrates "Query by Playing on a MIDI keyboard" (QbP), a general music retrieval approach that can take a MIDI sequence played on any instrument and *interactively* find and play similar music out

2375-026X/19/$31.00 ©2019 IEEE
DOI 10.1109/ICDE.2019.00223

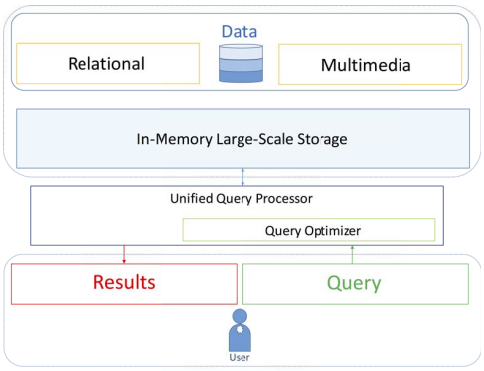

Fig. 1. Hybrid.poly architecture

of millions of music files available in a large-scale repository.

II. Architecture

The HYBRID.POLY architecture is illustrated in Figure 1. The *in-memory* storage engine is capable of the ingesting and query processing of data in different data models (e.g. relational, JSON, XML, MIDI, Video, etc). The query interface accepts user queries in a *hybrid* language that is a superset of SQL providing extra capabilities to support complex analytical workloads on both relational and non-relational data (e.g. media files) [26].

Query Language: With our *hybrid* SQL language, it is possible to store and query *vectors* and *matrices* as first-class objects. Without these new *data types*, we would have to resort to inefficient workarounds to convert these abstractions to their relational equivalent and back, as demonstrated below.

First, we store *vectors* as attributes of a new type *vector* of a regular relational table. Second, we embed several linear algebra operations into our hybrid linear-relational query language to simplify implementation of both in-database relational and linear-algebra workloads. For instance, let us consider computing a *dot product* of 2 vectors. In our hybrid SQL, the query looks very natural and concise:

```
SELECT id, vector1.dot(vector2) FROM VectorStorage
```

This is also possible in standard SQL, but requires the inefficient and more complex query with a GROUP BY and JOIN. Before that can occur, the vectors must be converted to their relational equivalent (i.e. value1, value2 that are their euclidean coordinates in the query below).

```
SELECT id, SUM(value1 * value2) FROM RelVecStore
WHERE index1 = index2 GROUP BY id
```

We added the following operators to extend SQL to support analytical workloads:

- **Plus**, **Minus**, **Dot**, **Equals** operators for calculating the sum, difference, dot product and equality of two vectors;
- **getData**, **setData** operators for accessing and modifying vector elements by index;
- **MusicMatch** operator for calculating the similarity between two note sequences;
- **SUM** aggregate operator extended from computing the aggregate on scalars to computing on vectors.

Using our hybrid SQL query language, it is easy to implement analytical workloads using both relational and analytical operators inside a relational engine without the need to export data for analytical processing outside the engine. We describe below how it can be done for Data Fusion, Machine learning, and Music search by writing concise hybrid SQL queries.

Although the same analytical workloads could be implemented with user-defined functions (UDFs), we modify the core language (add new abstractions and operators) to accomplish this task. There are several reasons why we have chosen this option. First, the newly introduced abstractions and operators are *reusable* among the many different analytical workloads, whereas a single UDF (e.g. for training a machine learning classifier) is usable only for this specific task. Second, a UDF is too abstract to participate in meaningful cost-based query optimization because it can be *any* user defined function, which severely limits optimization horizons. By contrast, it is possible to make the query optimizer aware of the new built-in types and operators and to design new semantic rules for the cost-based optimization that account for the semantics of these new data types and operators. For these reasons, it is more robust to add the new types to SQL than to have a custom UDF on an as-needed basis.

Data Model: We extend the *relational* data model with additional types and operations to support *matrix* and *vector* abstractions and *operators* over them. *Music* data types are stored in a relational model with additional metadata attributes and vectorized notes. With HYBRID.POLY, the user is also capable of ingesting and querying JSON documents when parsed by a JSON parser and stored in a relational model as a tree.

Storage Engine: In HYBRID.POLY we employ a distributed in-memory storage engine with optional *persistence*. The *distributed* storage handles large volumes of data while the *In-memory* storage decreases latency and enables *interactive* performance of complex analytical queries. *Persistence* provides fault tolerance in case of node failures or power outages. Finally, we were able to introduce support of different models on top of this storage, allowing the ingestion, processing, and querying of *heterogeneous* data.

Scalability: We run our scenarios on web-scale datasets by leveraging horizontal partitioning.

Query Optimization: We are currently working on a scalable hybrid query optimizer supporting both linear, and relational algebras, and non-linear operator nodes in the query tree.

III. User Scenarios

Interactive Music Search: The problem of music similarity search will often appear in real-world settings. For example, a musician with a new tune in mind might want to check if there are similar songs in order to avoid copyright infringement. Another musician may search for similar songs, just like any other researcher to become familiar with the latest

ideas and to find inspiration for a new song. Using our system and hybrid SQL language, it is possible to run *interactive* analytical workloads to retrieve and play similar music. We illustrate below how anyone interested can play a melody on a MIDI keyboard (or pick a tune from the existing database) and the system will interactively find similar music fragments. We store two music data sources – the original music MIDI files represented as records with two attributes (*notes* and *comment*) taken from the *Clean MIDI* subset of the Lakh Dataset [27], [28] and MIDI fragments played by the user, represented as records with the same two attributes.

```
SELECT r.score, r.comment FROM (
 SELECT m.vec_notes.music_match(i.vec_notes,
     'tone-invariant') AS score, m.comment
 FROM MidiStore AS m, MidiInput AS i) AS r
WHERE r.score > 0.98
```

After the user plays a melody (for example, a few notes from *Strangers in the Night* by *Frank Sinatra*), the notes are vectorized and ingested into our engine. The **musicMatch** operator is our extension to SQL that allows us to match two melodies by similarity and outputs a floating-point number in the $[0, 1]$ range. Once a certain similarity threshold has been reached, the attributes of a matching melody are printed out. The results are shown in Figure 2, which are displayed and played in real-time, so the user can enjoy a variety of similar music found by the system. As shown in the result, there are 16 tracks that fall under the criteria specified in the query. Of these 16 tracks, 11 are perfect matches and 5 are almost-perfect matches. There is a perfect match of the original track (line 1, *Sinatra Strangers in the Night*), a cover of the original track (line 2, *Kaempfert Strangers in the Night*), and a few seemingly unrelated melodies that contain the sequence of notes that is transpositionally equivalent to our fragment (enabled by *tone-invariant* keyword in the query). The most amusing and non-obvious match occurs in line 3, *Bach Johann Sebastian Orchestersuite Nr. 2 5. Polonaise*). As we listened to the rest of the listed tracks, we found the perfectly matched line 4, *Frank Sinatra Not as a Stranger*, is a mislabeled original track.

Machine Learning: As there are millions of web pages, web searching is an important problem, and without a search-engine it is very hard to find the required information. A web crawler is a critical component of a Web search engine that aims to browse all available web pages and create a repository for further processing, indexing, and searching. In this scenario, a stream of Web pages is produced by a Web crawler and is immediately classified into four different categories using a large-scale interactive multinomial Naive Bayes machine learning classifier [29] implemented as a *short query* in our language. The classification results can be used to improve the relevance of the Web search results.

The classifier is trained on a subset of a large-scale Web tables dataset [7] having millions of Web tables. Each document belongs to one of four classes: job postings, forum posts, songs and spam. There are a total of 21255 documents, 5000 of which are job postings, 5675 are forum posts, 4989 are

#	score	comment
1	1	Sinatra Strangers in the Night
2	1	Kaempfert Strangers in the Night
3	1	Bach Johann Sebastian Orchestersuite Nr. 2 5. Polonaise.6
4	1	Frank Sinatra Not as a Stranger
5	1	Soundgarden One Minute of Silence
6	1	Ocean Put Your Hand in the Hand
7	1	The Bangles Walk Like an Egyptian.4
8	1	The Beatles Day Tripper
9	1	Jamiroquai Cosmic Girl
10	1	Jamiroquai Cosmic Girl.1
11	1	Bryan Adams Summer of '69.2
12	0.98684	Rush Subdivisions.1
13	0.98342	Alliage Le temps qui court.1
14	0.98148	Eagles Hotel California.3
15	0.98095	Eric Clapton Bad Love
16	0.98058	Nine Inch Nails Wish

Fig. 2. The interactive music search result. The *score* is the measure of similarity between the melody played by the user and the music in the database. The *comment* is the metadata found in the MIDI file. A few melodies matched our played tune; among these matches were the original melody, a cover by *Kaemphert* and a few seemingly unrelated melodies such as the *Polonaise* by J.S. Bach. The detailed analysis of the results revealed the *Not as a Stranger* melody was mislabeled and is the original tune *Strangers in the Night*.

document	category
"new medical receptionist …"	job
"… la melodia del cielo 3:57 £0.79 view in itunes"	song
" top "	spam
"new environmental restoration engineer …"	job
"jazz forum desert island albums 433 6 hours ago"	post
"… started by craig morris via web …"	post
"… manager needed in naperville il $150000 …"	job
"our most noble cause should be…"	song

Fig. 3. A sample of the classification results from the large-scale machine learning demo scenario. The first six rows illustrate correctly classified instances. The last two rows are misclassifications.

songs and 5591 are spam. The training data is of the same format as the labeled tuples in Figure 3.

```
SELECT input.vec_wordCounts.dot(model.vec_weights)
  AS likelihood,
  model.category, input.id
 FROM Input AS input, Model AS model
 INTO Likelihood;
SELECT MAX(likelihood), id
 FROM Likelihood
 INTO MaxLikelihood
 GROUP BY id;
SELECT l.id, l.category
 FROM Likelihood AS l, MaxLikelihood AS ml
 WHERE l.likelihood = ml.likelihood
 AND l.id = ml.id;
```

We demonstrate the workload by asking a user to submit the queries in a shell, *interactively* observe the training and classification, and examine the classification results. As a result of classification, each document in the stream produced by a Web crawler is assigned the category predicted by the classifier. Figure 3 illustrates a sample of classification instances.

The listing above contains the queries that are used to

classify Web pages. The trained classifier is stored in table *Model*. For each of the categories a vector (i.e. a unidimensional array of doubles) representing that category is stored. The query training the model is omitted here. Given the input document's bag-of-word representation, a measure of similarity is computed for this document and each category. The category with the highest score is then assigned as a class label.

Evaluation: In order to evaluate our classifier, we have performed 10-fold cross-validation on our input dataset. The F measure per class is shown in Table I.

	posts	jobs	songs	spam
F measure	0.728	0.743	0.737	0.843

TABLE I
THE F MEASURE FOR EACH CLASS.

Classification of a bulk of 2126 documents takes a few seconds on a single Intel multi-core node and 192 GB RAM. Using horizontal partitioning, we can classify larger datasets by adding more machines to the cluster with our engine.

Music Search + Machine Learning + Data Fusion: In this final and most advanced scenario, we show that the workflows demonstrated in previous scenarios are composable. This important property means that we can integrate the time-consuming Extract-Transform-Load (ETL) procedures in the query, automate it once, thus avoiding them in future composite workflows. We illustrate this by composing the data integration and machine learning capabilities in the same query. The user can run the Songs classification on the entire Web tables dataset [7]. The web table rows which are classified as Songs are now fused with the MIDI tracks using the **musicMatch** operator and song title matching the MIDI metadata.

To simultaneously perform classification and data integration, the user can execute the following composite Data Fusion/Classification query. (Parts of the query have been omitted.) The results interactively follow the query submission.

```
SELECT wtsongs.allColumnsJoined
FROM (SELECT wt.allColumnsJoined, classified.label,
    wt.rowNum, wt.column1, wt.column2, ...,
    wt.column29
 FROM WebTables AS wt, (...) AS classified
 WHERE wt.id = classified.id) AS wtsongs, MidiStore
     AS ms, MidiInput AS mi
WHERE wtsongs.label = 1
AND mi.vec_notes.music_match(ms.vec_notes) > 0.95
AND (ms.comment.contains(wtsongs.column1) ...
OR ms.comment.contains(wtsongs.column29))
```

Looking at the results, the user discovers that *music similar to his/her taste exists on iTunes*. The user also discovers the price and the option to purchase the track or entire album. From the *technology perspective*, this query fragment incorporates all logic - *complex non-linear analytics (music matching), linear-algebra-based analytics (machine learning), and data fusion at scale - in a single hybrid SQL query*.

result
4 supermassive black hole … 3:29 $1.29 view in itunes …
6 supermassive black hole … 3:29 $1.29 view in itunes …

Fig. 4. The music search combined with machine learning and data integration. The table contains the songs from Web tables dataset similar to what the user plays. We can see that fusion with Web tables leads us to iTunes to purchase similar music.

REFERENCES

[1] M. Stonebraker, "Big data means at least three different things..." in *NIST Big Data Workshop*, 2012.
[2] M. Gubanov, "Polyfuse: A large-scale hybrid data fusion system," in *ICDE*, 2017.
[3] M. Podkorytov, D. Soderman, and M. Gubanov, "Hybrid.poly: An interactive large-scale in-memory analytical polystore," in *ICDMW*, 2017.
[4] M. Gubanov and M. Stonebraker, "Large-scale semantic profile extraction," in *EDBT*, 2014.
[5] B. Alexe, M. Gubanov *et al.*, "Simplifying information integration: Object-based flow-of-mappings framework for integration," in *BIRTE*, 2008.
[6] M. Gubanov and A. Pyayt, "READFAST: high-relevance search-engine for big text," in *CIKM*, 2013.
[7] M. Gubanov, M. Priya, and M. Podkorytov, "Cognitivedb: An intelligent navigator for large-scale dark structured data," in *WWW*, 2017.
[8] L. Shangyu, G. Zekai *et al.*, "Scalable linear algebra on a relational database system," in *ICDE*, 2017.
[9] M. Gubanov and P. A. Bernstein, "Structural text search and comparison using automatically extracted schema," in *WebDB*, 2006.
[10] M. Gubanov, L. Popa *et al.*, "Ibm ufo repository," *PVLDB*, 2009.
[11] M. Gubanov and L. G. Shapiro, "Using unified famous objects (ufo) to automate alzheimer's disease diagnostics," *BIBMW*, 2011.
[12] M. Gubanov, L. G. Shapiro, and A. Pyayt, "Learning unified famous objects (ufo) to bootstrap information integration," *IRI*, 2011.
[13] M. Gubanov, A. Pyayt, and L. G. Shapiro, "Readfast: Browsing large documents through unified famous objects (ufo)," *IRI*, 2011.
[14] M. Gubanov and A. Pyayt, "Medreadfast: A structural information retrieval engine for big clinical text," *IRI*, 2012.
[15] S. Cheemalapati, M. Gubanov *et al.*, "A real-time classification algorithm for emotion detection using portable eeg," *IRI*, 2013.
[16] M. Gubanov, M. Stonebraker, and D. Bruckner, "Text and structured data fusion in data tamer at scale," *ICDE*, 2014.
[17] Z. Abedjan, J. Morcos *et al.*, "Dataxformer: Leveraging the web for semantic transformations," in *CIDR*, 2015.
[18] M. Gubanov, P. A. Bernstein, and A. Moshchuk, "Model management engine for data integration with reverse-engineering support," in *ICDE*, 2008.
[19] R. Khan and M. Gubanov, "Nested dolls: Towards unsupervised clustering of web tables," in *Big Data*, 2018.
[20] M. Podkorytov and M. Gubanov, "Hybrid.poly: Performance evaluation of linear algebra analytical extensions," in *Big Data*, 2018.
[21] S. Soderman, A. Kola *et al.*, "Hybrid.ai: A learning search engine for large-scale structured data," in *WWW*, 2018.
[22] M. Boehm, M. W. Dusenberry *et al.*, "Systemml: Declarative machine learning on spark," in *VLDB*, 2016.
[23] T. Elgamal, S. Luo *et al.*, "Spoof: Sum-product optimization and operator fusion for large-scale machine learning," in *CIDR*, 2017.
[24] V. Gadepally, P. Chen *et al.*, "The bigdawg polystore system and architecture," in *HPEC*, 2016.
[25] Y. Zhu and D. Shasha, "Query by humming: a time series database approach," in *SIGMOD*, 2003.
[26] E. F. Codd, "A relational model of data for large shared data banks," in *CACM*, 1970.
[27] C. Raffel, "Learning-based methods for comparing sequences, with applications to audio-to-midi alignment and matching," in *PhD Thesis*, 2016.
[28] "online: http://colinraffel.com/projects/lmd/," 2016.
[29] J. D. M. Rennie, L. Shih *et al.*, "Tackling the poor assumptions of naive bayes text classifiers," in *ICML*, 2003.