PolyFuse: A Large-scale Hybrid Data Fusion System

Michael Gubanov
University of Texas at San Antonio

I. Introduction

The big data era brought us petabytes of data together with the challenges of storing and efficiently accessing large-scale datasets. However, it unexpectedly surprised everyone with an enormous *variety* of data sources and types, and corresponding different *data models*. Dealing with a variety of those data models turned out to be a "hard nut to crack" for almost all existing data management engines. *Data integration*, a mature field addressing problems of accessing and fusing data residing in more than one datasource, over the years came up with feasible semi-automatic solutions, most of which efficiently handle a handful of data sources represented in one or two different data models (e.g. relational and semi-structured) [Haas et al., 2005], [Gubanov et al., 2008]. While this is significant progress, most of the solutions do not easily scale up, since they usually require some sort of human assistance, infeasible at scale. *Unified Famous Object (UFO)* [Gubanov et al., 2009], [Bellahsene et al., 2011], [Gubanov et al., 2011a] was one of the first attempts to crack data fusion at scale by introducing a new abstraction called `UFO` that could incrementally learn different representations of data objects in different sources, and, over time, train itself to recognize and map them with high accuracy without supervision. Having trained many such `UFOs`, the system would get more and more powerful, as it learned to recognize and access data objects across many sources without supervision. While `UFO` was definitely progress towards scaling up data fusion, it was not a "silver bullet", because it was built mostly for relational data. Hence, there is a need for a new large-scale data integration system that could handle many types of data at scale. This paper sketches its architecture, and envisions potential research challenges. First, a few key *principles* that such a system should follow are described, then follows the discussion on architecture and research avenues.

Data Acquisition: Ingest *any* incoming data for further classification and processing

No Schema: Incoming data does not need to fit a pre-defined schema. Instead the schema is *extracted* from data if needed

Scale: Designed to accommodate and process large-scale heterogeneous data sets. For example, a fire hose of high-velocity sensor data or unstructured discharge reports from a hospital

Seamless Data Fusion: Unsupervised, source-oblivious fusion at scale to satisfy the user query

"Esperanto" Query Language: A hybrid query language enabling access to different data models used by heterogeneous data. It might be a result of partial composition of several query languages or a new language designed based on a common data model.

The first principle sets PolyFuse apart from a traditional group of DBMSes that were built for relational data models and were around for decades [Stonebraker et al., 1996]. The second principle removes a traditional requirement of data to "obey" a pre-defined schema in order to get into a database. The third principle means the architecture should be designed to support very large heterogeneous datasets from the very beginning. The fourth and fifth principles indicate that due to scale, any fusion algorithms should be *unsupervised* and any query language should be compatible with different *data models* in PolyFuse.

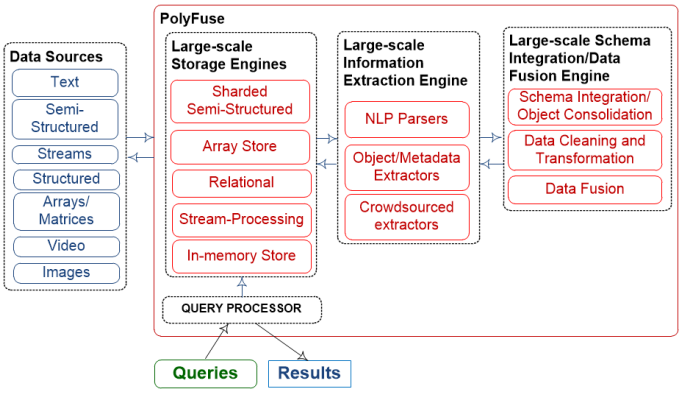

Fig. 1. Architecture

II. Architecture

The PolyFuse candidate architecture is depicted in Figure 1. The incoming data are automatically classified by type and ingested by the corresponding datastore that handles a data type. The user queries are processed by the `Query Processor`. It takes a query and answers it by using previously fused and cached data in the in-memory store, if such data is fully or partially unavailable, it reformulates the query into several queries, possibly in different languages, and submits them to the original data stores referenced in the query. Next, let us discuss the components from Figure 1 in more detail.

Large-scale Storage Engines/Raw Stores These data stores ingest and store original, raw data from incoming data sources, after their classification by type.

Relational: This is a classic relational row-store (e.g. PostgreSQL [pos,]) or a column-store (e.g. Vertica, C-Store [ver,], [Stonebraker et al., 2005]) that is used to store, index, and query structured data with SQL [Codd, 1983]

Array Store: This is a data management system to store, index, and query high-dimensional matrices and vectors (e.g. SciDB [sci,]). It is called an *array*-store because it uses an array data model to represent matrices and vectors, and executes linear-algebra operators on these data (e.g. matrix-vector multiplication, see [sci,]). However, it also can be an engine, supporting linear algebra via another data model, not necessarily arrays. For example, Hybrid [Gubanov et al., 2016], [Gubanov, 2017] supports matrices

and vectors as attributes of relational tables and allows native in-SQL support of linear algebra operators.

Stream Processing: This is a large-scale stream processing engine (e.g. Borealis [Abadi et al., 2005]) that processes streaming data and queries over them. For example, a data stream might come from a heart-lung machine carrying real-time information about blood oxygenation during heart surgery. The query on such data could perform a real-time Fourier transform to generate an alert if the patient is in danger.

Semi-Structured: This is a sharded JSON store or a distributed data management system with native XML and XQuery support (e.g. IBM DB2 pureXML [db2,]). These can be used to store, index, and access big volumes of semi-structured data. Unstructured data, such as text from a Web page, can be stored as well, after being parsed by a domain-dependent or domain-independent Natural Language Processing (NLP) parser [Gubanov and Bernstein, 2006]. A parser should be capable of identifying and extracting objects, their properties, and relationships from text. After that, the original text is no longer unstructured, because it is enriched by this new extracted metadata. It can be stored in a semi-structured store and accessed via a semi-structured language, such as JSON or XQuery. Similarly, other types of unstructured data, such as images, video, binary data streams, can be enriched with metadata using a custom metadata extractor, and stored in a semi-structured store [Gubanov, 2017].

In-memory store: This is a large-scale in-memory engine (e.g. VoltDB, MemSQL, Spark [mem,], [Zaharia et al., 2010]) that stores global integrated schemas, fused by the large-scale schema integration and data fusion module described below.

Large-scale Information Extraction Engine

NLP Parser can be domain-dependent or domain-independent. The domain-dependent parser processes text and automatically identifies and tags a fixed set of objects/entities from that domain. For example, a medication tagger would parse Electronic Health Records (EHR) and tag all medications.

Deep or shallow Natural Language Processing (NLP) parsers [Klein and Manning, 2007] can enrich plain grammatical text with certain tags, indicating Part-Of-Speech, Part-Of-Sentence, and other linguistic information. It can be used to develop or train a domain-independent parser that can identify objects in text, just like a domain-dependent parser does for a fixed domain, but now without a domain restriction.

In both cases, unstructured raw text is enriched with additional metadata (tags), which turns it into semi-structured data. It can be stored in a semi-structured store described above and accessed using JSON or XQuery query languages that will leverage additional metadata. For example, a user would be able to retrieve all countries from text, which is impossible without one of these parsers.

Metadata extractor can be software that reads metadata attached to an image or video file, or processes a video file to generate a script of what is being discussed in the video.

Crowdsourced extractor can use crowdsourcing to extract metadata that is otherwise impossible to extract.

Large-scale Schema Integration/Data Fusion Engine

Schema Integration: This module is responsible for schema integration of metadata already present in structured data or extracted from unstructured data [Priya et al., 2017]. It includes consolidation and unification of schemas [Gubanov et al., 2008], [Haas et al., 2005], [Melnik et al., 2003], [Bollacker et al., 2008], before sending them to the query processor, so the user can query the global integrated schema [Gubanov et al., 2009], [Gubanov et al., 2011b], [Gubanov et al., 2008].

Transformation/Cleaning: Data cleaning is essential when handling large-scale data [Chu et al., 2015]. Text, tables, or any other data that are created by humans naturally contain misspellings, refer to the same objects differently, and have inconsistencies. For example, `Mcrsoft` is a misspelling of `Microsoft` and a data cleaner should be able to recognize and associate `Mcrsoft` with a correct data object automatically.

Query Processor accepts a user query and processes it via an in-memory store with cached data and an integrated global schema. If this is not possible, it reformulates and forwards the query in whole or in part to the raw stores. Then, it merges responses from different engines to output a fused query result. This component is conceptually similar to mediators in P2P data management, although developed mostly for structured data [Tatarinov et al., 2003].

III. Related Work

Polystores are relevant systems in that they also aim to store different kinds of data in a distributed fashion. Our focus here, however, is on significant challenges of *data fusion*, *information extraction*, *data cleaning*, given a variety of large-scale data sources in the big data era. Polystores also mention these research areas, but nevertheless are much more focused on high performance query processing and query optimization questions in a distributed setting, rather than on heterogeneity of large-scale data and consequent challenges. The work on distributed databases is also relevant to PolyFuse. Federated relational databases were around for decades and were designed to store and query distributed *structured* data [Stonebraker et al., 1996], [Haas et al., 2002], [Haas et al., 2005].

Much research addressing *variety* of data was historically done by the data integration research community. It is a rich research area with significant research contributions [Melnik et al., 2003], [Haas et al., 2005], [Haas et al., 2002], [Madhavan et al., 2005]. While this work represents significant progress, most of the solutions focus only on *structured* or *semi-structured* data, and most of them require a human in the loop, which is infeasible for a large number of data sources. Dataspaces introduced a *pay-as-you-go* data integration approach that postponed labor intensive integration until absolutely needed (e.g. at query time), but similarly suffer from such problems and lack large-scale adoption.

UFO Repository [Gubanov et al., 2009] was one of the first unsupervised solutions to challenge large-scale data integration in structured and semi-structured data worlds. It introduced a higher-level abstraction, the `Unified Famous Object (UFO)`, and leveraged it to automate and scale data integration. In the same way that a Java Object hides *implementation* details behind its interface, a Unified Famous Object conceals data representation differences in different data sources. A large collection of `UFOs` simplifies data access and integration by automatically recognizing objects in the incoming data

feeds and offering a standard query interface oblivious of the source schemas. Finally, UFOs are more general and flexible than schemas in the sense that they can be viewed as abstract building blocks for metadata-intensive applications.

There are several important properties that distinguish UFO repository from other data integration solutions. Neither data warehousing, ETL, nor schema or ontology matching [Hernández et al., 2008], [Rahm and Bernstein, 2001], [Madhavan et al., 2005] attempt to introduce a common abstraction and leverage it to raise the level of automation. In some sense, those approaches are less general in that they put main efforts on the specifics of data mapping and transformation and are built for *specific* query languages, schemas, ontologies, or data formats. In contrast, UFO crystallizes a higher-level abstraction that conceals the lower-level representation details.

Compared to UFOs, traditional schemas lack *modularity*, *standardization*, ability to automatically learn from data, and *reusability*. This difference is substantial and can be compared to a difference between *functional* and *object-oriented* programming. UFO, as one of the scalable data integration technologies, can be used inside the data integration component of POLYFUSE.

IV. RESEARCH CHALLENGES

Information Extraction

Scalable extractors and taggers are needed to identify and extract objects from unstructured data, such as text, images, and video. For example, for plain text, this task is often performed by a domain expert, having sufficient knowledge of the area, in order to identify and correctly tag entities and their types. The discussion below relates mostly to plain text, however, some of the challenges are very similar for video and images.

Domain-dependent information extraction is concerned with the problem of tagging entities, their properties, and relationships in a document or data file from a specific domain. For example, for discharge reports from a hospital of a patient who underwent cancer therapy, a nurse would be able to read the report and identify/tag the *medications* used for treatment. The same nurse would most likely be unable to do the same task of entity tagging in a different domain, unless she happens to have sufficient domain knowledge. Even within the same medical domain, outside the cancer sub-domain the accuracy of her tagging might degrade. This example demonstrates that on a small scale, within the same domain, the task of information extraction can be performed by a human. A human can be substituted by a domain-dependent extractor, usually a machine-learning classifier, trained to identify a specific entity type, for example, medications. Accuracy of such a tagger can be sufficiently high to substitute a human (more than 95% [Gubanov and Stonebraker, 2013]). However, the more entities are to be tagged, the more taggers need to be trained. For example, [Gubanov and Stonebraker, 2014] uses a semi-structured dataset that was a result of applying more than 100 different entity-taggers. It is a rich dataset, but 100 taggers are expensive, challenging, and sometimes infeasible to train. Hence, to scale up in order to identify hundreds of different entities or process information from different domains, domain-independent information extraction might be a viable option.

Domain-independent information extraction is concerned with the problem of identifying entities, their properties, and relationships in a more general way, compared to the domain-dependent approach. Rather than training a tagger for every entity in a domain, a domain-independent parser parses natural language sentences into an intermediate semi-structured representation that can be further used to define a small template for an entity tagger regardless of the domain. For example, a dependency tree [Klein and Manning, 2007] or shallow parse are examples of an intermediate representation that eases the burden of training a tagger per entity, as it is not tied to any domain. One of the challenges in domain-independent information extraction is the inaccuracy of NLP parsers needed to produce such intermediate representations [Klein and Manning, 2007]. It leads to an incorrect parse, which in turn can significantly reduce precision and recall of an entity tagger. In addition to that an NLP parser usually works only on grammatical text, which makes it inapplicable to any text without a correct language grammar.

There is controversy regarding strengths and weaknesses of both approaches and there is still no scalable, fully unsupervised solution available that can be applied to extract entities, attributes, and relationships from a large-scale grammatical or non-grammatical text from any domain.

Schema Integration, Data Cleaning/Fusion

Since data are large-scale and the number of sources is large (e.g. millions of tables from the Web), this module should be fully automatic, which means avoiding human intervention. This requirement makes many well-known existing schema integration solutions unsuitable, because most of them are semi-automatic and require human assistance during the integration process. *Unified Famous Objects* [Gubanov et al., 2009] made some progress to scale up schema integration, while minimizing or in some cases completely bypassing human intervention, although mostly for structured data. The challenge still remains open to support more data types.

Similarly, *Data transformation, Cleaning, and Fusion* at scale need to support a variety of data types, yet remain unsupervised. [Abedjan et al., 2014] is a recent work in this direction that leverages the Web to automate the data transformation process for structured data. The challenge still remains to support more data sources as well as types of data.

Query Processing

Data in POLYFUSE can be accessed through the *Query Processor*. The author envisions a query processor capable of processing keyword-queries, SQL, XQuery, and JSON depending on the data used by the system and the query language preferred by the user. Also a query language supporting *hybrid* queries in two or more languages is needed, if the data is heterogeneous and there is no wrapper available from one dataset to another dataset.

For example, assume the user is mostly working with two types of data - relational and semi-structured, and expects the query result to be a relational table. She would then ingest her files with a hope to fuse them and be able to query both of them at once. She could manually specify a `SQL:` prefix in the query portion accessing relational data and `JSON:` prefix in the part querying semi-structured JSON data. Below, a *hybrid* query, fusing information from Web text (semi-structured) and Web tables (relational) [Cafarella et al., 2008] data sources is illustrated. The query outputs the best deal for a popular Broadway show.

```
SELECT MIN(S.Price) AS Price, S.Show, S.Theater,
       S.ShowTime
FROM Shows AS S
WHERE S.Show IN
  (SELECT WebText.TITLE
  FROM
  CAST(WebText,
       JSON:
       db.find({
         "WebText.location":"Broadway",
         "WebText.type":"Show",
         "WebText.html":/amazing/})
    }))
GROUP BY S.Show, S.Theater, S.ShowTime
ORDER BY Price ASC
```

First, it would find all shows on Broadway mentioned on Social Web (stored in a JSON database `WebText`), having keyword *amazing* in the user comments[1]. Next, it would access data from a relational table `Shows`, crawled from the Web, having prices, theater addresses, and show times to find the best deal.

In homogeneous (one-language) queries, the language can be detected automatically by the `Query Processor` using a machine learning classifier [Gubanov and Pyayt, 2016] trained to detect the query language.

Query Optimization

Traditional query optimization is not directly applicable to POLYFUSE. First, in order to gauge total execution cost, the optimizer would have to have precise implementation models of all operators from all storage engines. For example, a JSON join is usually a distributed *nested-loop* join, in an array store it might be a version of *scatter and gather* and it is not the same in different array stores. Hence, adding a new engine to such a system would require updating the optimizer with models of all new operators, which would prohibitively complicate each addition. Second, traditional cost-based optimizers heavily depend on statistics (histograms for the value distribution of entity attributes). Such statistics might not be exposed by all stores, or it may not even be supported. Hence, a more flexible and general query optimization strategy is needed for a large-scale hybrid data integration and query processing system like POLYFUSE.

Acknowledgments: The author would like to thank anonymous reviewers, Michael Stonebraker, Manju Priya, and Maksim Podkorytov for their comments and discussions of the earlier versions of this work.

REFERENCES

[sci,] online: http://scidb.org.

[db2,] online: http://www.ibm.com/db2/xml.

[mem,] online: http://www.memsql.com/.

[pos,] online: http://www.postgresql.org/.

[ver,] online: http://www.vertica.com.

[Abadi et al., 2005] Abadi, D. J., Ahmad, Y., Balazinska, M., Cetintemel, U., Cherniack, M., Hwang, J.-H., Lindner, W., Maskey, A. S., Rasin, A., Ryvkina, E., Tatbul, N., Xing, Y., and Zdonik, S. (2005). The Design of the Borealis Stream Processing Engine. In *CIDR)*.

[Abedjan et al., 2014] Abedjan, Z., Morcos, J., Gubanov, M., Ilyas, I., Stonebraker, M., Papotti, P., and Ouzzani, M. (2014). Dataxformer: Leveraging the web for semantic transformations. In *CIDR*.

[1]For example from a Twitter or Facebook feed, where users share their opinions about shows they recently attended

[Bellahsene et al., 2011] Bellahsene, Z., Bonifati, A., and Rahm, E. (2011). Schema matching and mapping. In *Springer*.

[Bollacker et al., 2008] Bollacker, K., Evans, C., Paritosh, P., Sturge, T., and Taylor, J. (2008). Freebase: a collaboratively created graph database for structuring human knowledge. In *SIGMOD*.

[Cafarella et al., 2008] Cafarella, M. J., Halevy, A., Wang, D. Z., Wu, E., and Zhang, Y. (2008). Webtables: exploring the power of tables on the web. In *VLDB*.

[Chu et al., 2015] Chu, X., Morcos, J., Ilyas, I. F., Ouzzani, M., Papotti, P., Tang, N., and Ye, Y. (2015). Katara: A data cleaning system powered by knowledge bases and crowdsourcing. In *SIGMOD*.

[Codd, 1983] Codd, E. (1983). A relational model of data for large shared data banks. *Commun. ACM*, 26(1):64–69.

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

[Gubanov and Bernstein, 2006] Gubanov, M. and Bernstein, P. A. (2006). Structural text search and comparison using automatically extracted schema. In *WebDB*.

[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., 2016] Gubanov, M., Jermaine, C., Gao, Z., and Luo, S. (2016). Hybrid: A large-scale linear-relational database management system. In *MIT Annual DB*.

[Gubanov et al., 2009] Gubanov, M., Popa, L., Ho, H., Pirahesh, H., Chang, P., and Chen, L. (2009). Ibm ufo repository. In *VLDB*.

[Gubanov and Pyayt, 2016] Gubanov, M. and Pyayt, A. (2016). Type-aware web search. In *EDBT*.

[Gubanov et al., 2011a] Gubanov, M., Pyayt, A., and Shapiro, L. (2011a). Readfast: Browsing large documents through unified famous objects (ufo). In *IRI*.

[Gubanov et al., 2011b] Gubanov, M., Shapiro, L., and Pyayt, A. (2011b). Learning unified famous objects (ufo) to bootstrap information integration. In *IRI*.

[Gubanov and Stonebraker, 2013] Gubanov, M. and Stonebraker, M. (2013). Bootstraping synonym resolution at web scale. In *DIMACS*.

[Gubanov and Stonebraker, 2014] Gubanov, M. and Stonebraker, M. (2014). Large-scale semantic profile extraction. In *EDBT*.

[Haas et al., 2005] Haas, L. M., Hernández, M. A., Ho, H., Popa, L., and Roth, M. (2005). Clio grows up: from research prototype to industrial tool. In *SIGMOD*.

[Haas et al., 2002] Haas, L. M., Lin, E. T., and Roth, M. A. (2002). Data integration through database federation. *IBM Syst. J.*, 41(4).

[Hernández et al., 2008] Hernández, M., Dessloch, S., and et al (2008). Orchid: Integrating schema mapping and etl. *ICDE*.

[Klein and Manning, 2007] Klein, D. and Manning, C. (2007). Fast exact inference with a factored model for natural language parsing.

[Madhavan et al., 2005] Madhavan, J., Bernstein, P. A., Doan, A., and Halevy, A. (2005). Corpus-based schema matching. In *ICDE*.

[Melnik et al., 2003] Melnik, S., Rahm, E., and Bernstein, P. A. (2003). Rondo: a programming platform for generic model management. In *SIGMOD*.

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

[Rahm and Bernstein, 2001] Rahm, E. and Bernstein, P. A. (2001). A survey of approaches to automatic schema matching. *VLDB JOURNAL*, 10:2001.

[Stonebraker et al., 2005] Stonebraker, M., Abadi, D. J., Batkin, A., Chen, X., Cherniack, M., Ferreira, M., Lau, E., Lin, A., Madden, S., O'Neil, E., O'Neil, P., Rasin, A., Tran, N., and Zdonik, S. (2005). C-store: A column-oriented dbms. In *VLDB*.

[Stonebraker et al., 1996] Stonebraker, M., Aoki, P. M., Litwin, W., Pfeffer, A., Sah, A., Sidell, J., Staelin, C., and Yu, A. (1996). Mariposa: A wide-area distributed database system. *VLDB Journal*, 5(1):48–63.

[Tatarinov et al., 2003] Tatarinov, I., Ives, Z., and et al (2003). The piazza peer data management project. *SIGMOD Rec.*, 32(3):47–52.

[Zaharia et al., 2010] Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., and Stoica, I. (2010). Spark: Cluster computing with working sets. In *Proceedings of the 2Nd USENIX Conference*, HotCloud.