Integrated Access to Big Data Polystores through a Knowledge-driven Framework

Justin McHugh, Paul E. Cuddihy, Jenny Weisenberg Williams,
Kareem S. Aggour, Vijay S. Kumar and Varish Mulwad
AI & Machine Learning – Knowledge Services & Big Data
GE Global Research
Niskayuna, NY 12309 USA
{mchugh, cuddihy, weisenje, aggour, v.kumar1, varish.mulwad}@ge.com

***Abstract*—The recent successes of commercial cognitive and AI applications have cast a spotlight on knowledge graphs and the benefits of consuming structured semantic data. Today, knowledge graphs are ubiquitous to the extent that organizations often view them as a "single source of truth" for all of their data and other digital artifacts. In most organizations, however, Big Data comes in many different forms including time series, images, and unstructured text, which often are not suitable for efficient storage within a knowledge graph. This paper presents the Semantics Toolkit (SemTK), a framework that enables access to polyglot-persistent Big Data stores while giving the appearance that all data is fully captured within a knowledge graph. SemTK allows data to be stored across multiple storage platforms (e.g., Big Data stores such as Hadoop, graph databases, and semantic triple stores) – with the best-suited platform adopted for each data type – while maintaining a single logical interface and point of access, thereby giving users a knowledge-driven veneer across their data. We describe the ease of use and benefits of constructing and querying polystore knowledge graphs with SemTK via four industrial use cases at GE.**

Keywords—semantic modeling, knowledge representation, big data, data integration, query processing

I. INTRODUCTION

Cognitive and AI applications including recommendation engines, chatbots, and intelligent assistants increasingly rely on domain knowledge captured in the form of structured semantic data. The recent commercial successes of applications such as Siri, Cortana, Google Now and Watson have fostered considerable interest in the construction and consumption of large-scale *knowledge graphs*. The domain of knowledge graphs can range from all information on the World Wide Web (e.g., Google Knowledge Graph [1], Microsoft Bing Satori [2]) to content restricted to within an enterprise (e.g., LinkedIn Knowledge Graph [3], Facebook Entity Graph [4]). Today, critical applications in many large-scale enterprises are powered by knowledge graphs comprising millions to billions of entities and hundreds of billions of associated facts and relationships.

With the maturation of Big Data and cloud computing giving enterprises access to more data than ever before, enterprises now need to address not only larger data volumes but also multimodal data—relational, graph, streaming, text, semi-structured, images, etc. Moving away from the "one size fits all" approach, alternate strategies for data storage such as polystores [5] or 'data lakes' that comprise multiple storage platforms (such as RDBMS, NoSQL, graph database, array database, HDFS) have become the norm in enterprises. Each dataset is persisted in its original form in the storage substrate most suited for its specific data model. This model leads to data environments becoming silos-of-silos, requiring the schema and access mechanisms of each silo to be well-known to effectively access the data. Consequently, traditional data access approaches lead to multiple, potentially unaligned models being written directly into systems and analytics, with multiple storage engines involved in serving a single data access request.

Several well-known open and commercial knowledge graphs demonstrate the need to integrate semantic and non-semantic Big Data. However, at present, there is a lack of standardization in both the definition of what constitutes knowledge in a knowledge graph and in the technology stacks used to construct and maintain knowledge graphs, including a lack of unity with respect to integrating knowledge graphs with non-semantic data stores. Open, cross-domain web-scale knowledge bases (e.g., Wikidata [6], DBpedia [7] and YAGO [8]) have in common an underlying technology stack built using Semantic Web W3C standards[1] but differ in terms of their data sources and knowledge extraction techniques. In such 'traditional' knowledge graphs, the domain model describing entities and their relationships as well as all conforming instance data is captured in a semantic store – typically a triple store or graph database.

While the technical approaches to maintaining commercial cross-domain web-scale knowledge graphs (e.g. Google Knowledge Graph [1], Microsoft Bing Satori [2], Yahoo! Knowledge Graph [9], Baidu Zhixin) as well as enterprise knowledge graphs (e.g., from LinkedIn, Facebook, Amazon, Walmart and various government agencies) have not been widely publicized, we fathom, given the volume, velocity and variety of Big Data collections they work with, that such graphs comprise more than just a semantic store, and that knowledge graph storage and query techniques vary widely across organizations.

[1] https://www.w3.org/standards/semanticweb/

Table 1. Knowledge Graph technology stack components

Technology	Products/Vendors
RDF/triple stores (Semantic graphs)	AllegroGraph, Virtuoso, GraphDB, Rya, Stardog, MarkLogic, BlazeGraph, Cray Graph Engine, Oracle Spatial & Graph
Property graph databases	Neo4j, DataStax Enterprise Graph, InfiniteGraph, JanusGraph, OrientDB, Oracle Big Data Spatial & Graph
Semantic middleware	Cayley, DGraph, Grakn.ai, Leapsight Semantic Dataspace, Unigraph, PoolParty Semantic Suite

Organizations today have multiple options (many shown in Table 1) for building knowledge graphs; they can roll out their own custom knowledge graphs using one of many available triple stores or property graph databases. Alternatively, they can leverage semantic middleware solutions that provide service hooks needed to develop W3C-compliant knowledge graphs on top of these graph stores. Property graph databases are very efficient for storing large volumes of linked data and performing traditional graph operations such as finding cliques and shortest paths. On the other hand, semantic triple stores dramatically simplify the creation and use of knowledge graphs with diverse, often hierarchical classes. Moreover, the availability of standards for machine-readable data structured around a domain model provides many advantages including a mechanism for interrogating the domain model, the ability to perform inference, and the ability to apply several models to a dataset with unbounded numbers of classes and properties.

Whether utilizing a triple store or other technology, the existing solutions do not adequately address the need for knowledge graphs to enable integrated access to diverse polyglot-persistent data stores, which is extremely important to large and diverse organizations such as GE. Motivated by this lack of standard approaches and technologies for constructing and maintaining knowledge graphs over Big Data polystores, we developed the Semantics Toolkit (SemTK) in part to make data from diverse physical sources accessible and consumable through a single logical, knowledge-driven layer.

In this work, we subscribe to the proposed definition in [10], viewing a knowledge graph as an abstraction layer wherein: (i) information from multiple (potentially Big Data) data sources can be seamlessly integrated using ontologies (semantic models), and (ii) an implementation of the abstraction layer can allow applications to effectively consume linked data from these sources in accordance with semantic models. We extend the traditional knowledge graph model to additionally represent entities and relationships describing data stored in polystore systems external to the semantic store, as well as how to access that external data. The polystore system can comprise triple stores, property graph databases, time series databases (i.e., historians for storing high-velocity data), distributed file systems (for storing high-volume unstructured data), or spread out across a collection of federated data stores. Further, we have built novel services to automate the retrieval of instance data from these stores in the context of a domain model. Through these services, a user specifies data of interest – using well-understood domain terms – and SemTK determines how to retrieve that data. The user does not require any practical knowledge as to where and how data is stored. SemTK is designed to enable the modeling of diverse datasets and automate the retrieval and integration of Big Data in a manner transparent to the consumer. This elevates the model used to retrieve data from being implicit within a given retriever's codebase to a computable artifact that can be shared by multiple retrievers and kept up-to-date without alteration of the calling system.

The remainder of this paper is organized as follows: Section II describes the conceptual approach underpinning our work. Section III provides a detailed review of the Semantics Toolkit architecture. Section IV describes capabilities and systems successfully delivered at GE using SemTK. Section V outlines related work in merging semantics and Big Data, and Section VI provides conclusions and outlines future work.

II. Conceptual Approach

Knowledge graphs built on a Semantic Web technology stack have at their foundation a semantic domain model referred to as an ontology. The ontology defines the universe of entities in the domain, their structure, properties, and the relationships between them. Together, the domain model and instance data allows for data to be captured in a linked, computable form as a collection of triples within a semantic triple store. Semantic Web technologies are valuable for capturing information structured in domain model terms intuitive to domain experts. As such, interacting with the data requires no knowledge of a database schema or the underlying storage mechanism, lowering the barrier of entry for users to interact with knowledge bases, and enabling access by domain experts and software developers alike.

While powerful, many types of data in this era of Big Data are not well-suited for storage and processing within a semantic triple store. For example, image files, often sized from kilobytes to tens of gigabytes, cannot be efficiently captured in a triple store. Similarly, time series datasets would incur high overhead if captured in a triple store, while also sacrificing key features that exist in most historians such as optimizations for efficient access in chronological order and built-in operations such as time alignment, interpolation and aggregation. It would be ideal to continue to store different types of data wherever they are most efficiently kept, yet provide the capability to interact with them in domain model terms.

In traditional data storage systems, the responsibility of applying a domain model to understand the context of the data is placed on the caller. This can lead to the development of multiple, potentially divergent interpretations of the same data. The tendency for such context to be embedded directly within applications removes the ability for such interpretations to be directly compared and harmonized. Extending the semantic stack to allow domain model-based access to data external to a triple store would allow users to interact with the data in context, enabling more informed and consistent usage of the data.

SemTK furthers the objectives above by enabling the retrieval of Big Data based on semantically-defined characteristics capturing metadata for data stored externally to the triple store, such as how it was acquired, what it means, and how it links to other datasets. Describing data in this way allows data scientists, application programmers and analytics alike to describe their desired data contextually, while relying on the framework to maintain awareness of data locations and schemas. Further, SemTK simplifies the task of data retrieval, serving as a single logical interface that retrieves and fuses data from multiple underlying semantic and Big Data repositories. External stores are automatically queried, and the data post-processed and filtered based on constraints passed to the framework services by the caller. When operations are completed, partial results from various stores are merged, presenting a single result set that contains both the Big Data and the semantic context related to the records. In this way, SemTK furthers our vision of delivering the benefits of the Semantic Web to Big Data systems and applications.

III. Architecture

SemTK was designed to lower the barrier of creating and integrating knowledge-driven applications and services into critical workloads.

The core SemTK functionality provides capabilities to interact with a knowledge graph in a triple store, including ingesting, querying and manipulating data. As described above, a semantic triple store alone is often insufficient, as many types of data are best captured in Big Data stores. Thus, SemTK was extended to wrap external query services, providing ontology-driven access to non-semantic datasets. This integration is not limited to the retrieval of data, however—the same service wrapping approach can be used to execute Big Data processing tasks such as the launching of Apache Spark or Hadoop MapReduce jobs.

SemTK is designed to work equally well in monolithic applications, Big Data applications and network applications composed of collections of interacting microservices. SemTK functionality is contained in Java libraries, which are also made accessible via microservice wrappers.

A. *Example Walkthrough of SemTK Usage*

Assume a user wishes to retrieve time series data for a set of industrial assets meeting certain criteria, using SemTK's web interface (shown in Figure 1). First, the user identifies the desired set of assets by browsing an ontology and selecting fields of interest to the query. As the user builds the query, pathfinding is used behind the scenes to connect the fields of interest across the classes of the semantic model. The selected fields and links between them are then used to automatically generate queries against the semantic store. The user may choose to apply filters to any field of interest, with the valid filter choices automatically populated. The user continues to refine the query, interacting with the knowledge graph until the proper set of assets is identified. The user can then add the time series data field to the query, potentially specifying specific time series variables or time ranges to retrieve. When adding the time series fields, the user is simply selecting additional fields from the ontology, and need not be aware that the time series data is stored outside of the knowledge graph.

When the query is run, the system performs the semantic queries and any required external queries to gather the time series data. The result sets are merged into a single result set which combines the time series data with the contextual information from the triple store.

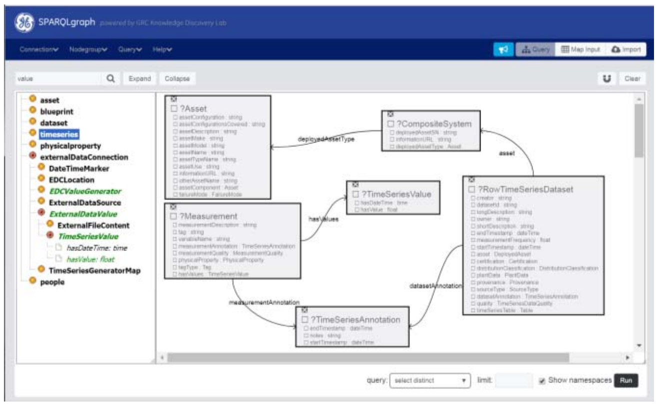

Figure 1. Semantic model visualized in SemTK's web interface

The next section describes the architectural components that enable this functionality.

B. *Functional Components*

1) *Nodegroup*

The nodegroup component is a stored representation of a subgraph needed to fulfill a user query. The nodegroup contains a set of classes from the ontology and the links between them, as well as the properties designated to be returned or constrained for each class. The nodegroup is also used to determine whether the query can be fulfilled solely by data in the semantic triple store, or whether it requires accessing a service external to the triple store. The nodegroup is a key data structure used pervasively within SemTK, and can be thought of as a SQL stored procedure-like capability.

2) *Ontology Info*

The Ontology Info component is a subset of the semantic model cached in memory, enabling efficient performance for critical query generation and validation operations. The Ontology Info object provides a hierarchical representation

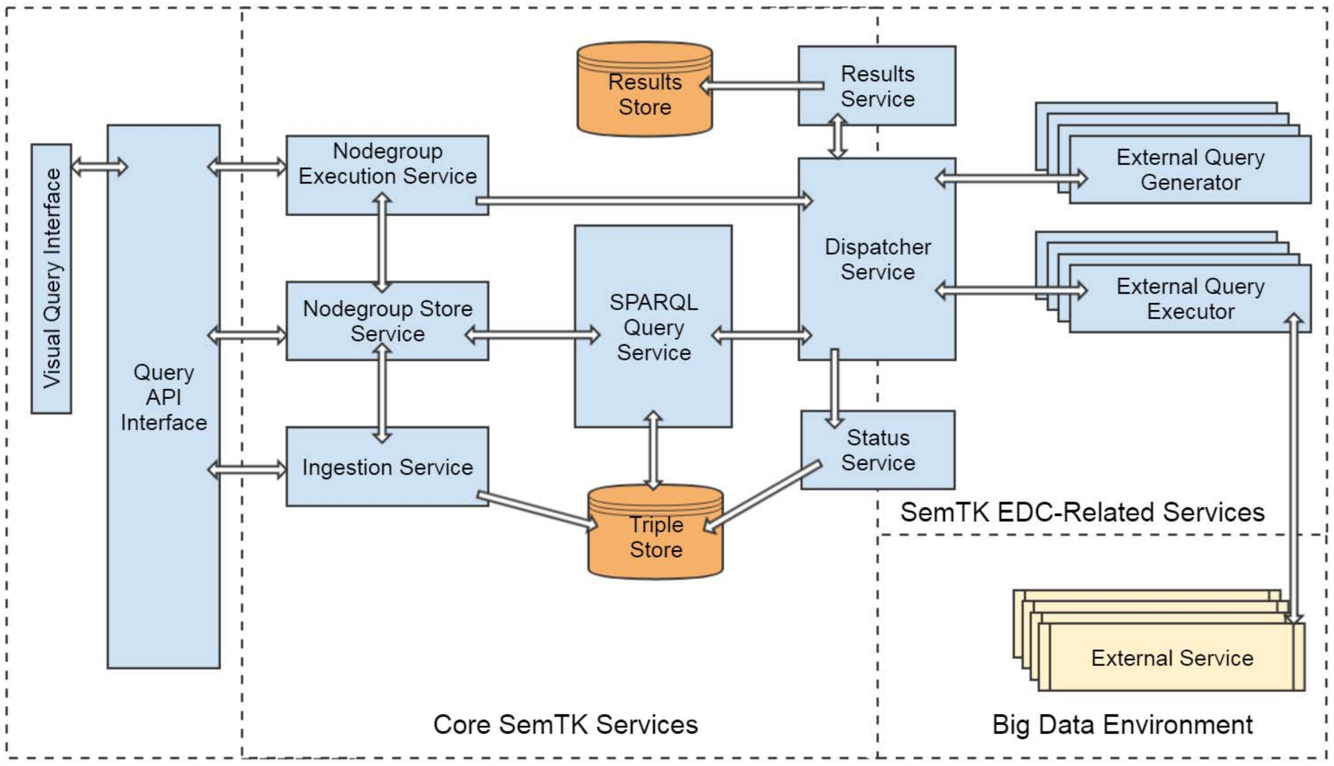

Figure 2. The SemTK with EDC reference implementation, connected to a user interface and Big Data environment

of the ontology, from which a user can select items when building a nodegroup. It also serves as a cache of the ontology, enabling efficient pathfinding when building nodegroups. The Ontology Info class and superclass information is used when examining a nodegroup to determine the presence of a class or relationship requiring invocation of an external service.

3) Pathfinding

Pathfinding finds connections between arbitrary fields of interest in the ontology. Pathfinding works by using a new class instance as the path endpoint and all the classes in the existing nodegroup as starting points. If needed, intervening classes are suggested as part of the potential paths. This feature is invoked in the user interfaces as users assemble nodegroups, and by the service layer when augmenting incoming nodegroups to include critical metadata required for accessing Big Data services.

C. Infrastructure Components

The reference implementation of the SemTK framework is constructed from a collection of microservices, shown in Figure 2, which communicate over HTTP.

1) Core SemTK Components

The Semantics Toolkit core components [11] provide common features for building, querying and managing knowledge graphs. These include services to perform queries, ingest data, store nodegroups for future use, and execute stored nodegroups. These features are used by most applications built with SemTK. There is also support for stored procedures, allowing SemTK queries to be integrated into applications and workflows without a deep understanding of semantic technologies.

2) External Data Connectivity (EDC) Components

The SemTK EDC components allow for interaction with external data services. Though these are conceptually treated as data stores, any service that accepts requests and generates return data (such as a Spark or Hadoop job) may be wrapped for invocation.

EDC functionality is driven by an EDC configuration, which is stored in a dedicated graph in the triple store. The EDC components manage the metadata regarding how to access the services. This includes information about available services, datatype-specific filtering opportunities, the type and location of external data related to a semantic model's instance data, and the required metadata to query a given external system.

Figure 3 shows a typical workflow to fulfill a user query. First, a nodegroup representing the user query is sent to the Dispatcher (Step 1). The Dispatcher checks the nodegroup for classes indicating that external services (EDC) must be used to retrieve non-semantic data (Steps 2, 3). Upon detection of an EDC trigger, the nodegroup is augmented to include any missing metadata required by the external service (Steps 4, 5), and the augmented nodegroup is returned for execution (Step 6). The Dispatcher performs the semantic portion of the queries and creates bins containing result subsets by grouping the largest subsets of shared

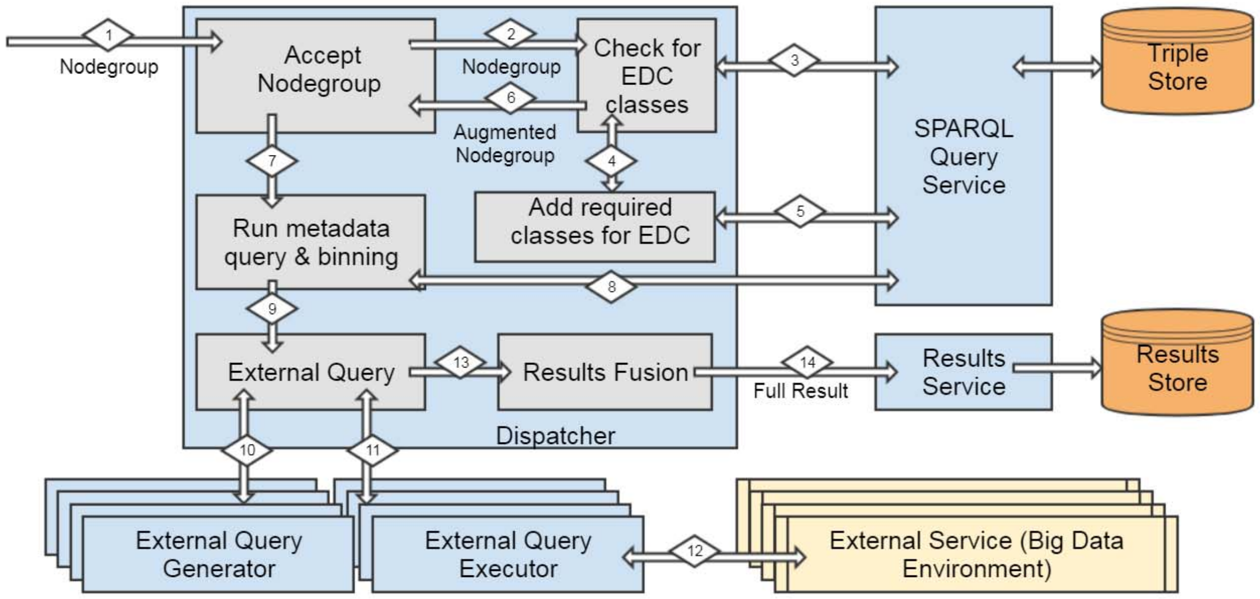

Figure 3. Operations relating to the Dispatcher, from query acceptance to results fusion

property values (Steps 7, 8). These bins are each given a UUID, used later in the fusion process. The external services are then triggered (Step 9). An External Query Generation service (Step 10) uses the bin IDs as a guide for the number of external queries to generate. A corresponding External Query Executor is then called to execute the generated queries in the Big Data environment (Steps 11, 12). Each partial result is annotated with the bin ID used to generate its query. The external query results and the semantic query results are fused (Step 13) using the bin IDs. Finally, the completed result set is sent to the Results Service (Step 14) for access by the calling UI or application.

a) Dispatcher

The dispatcher is responsible for orchestrating fulfillment of queries. This fulfillment is performed asynchronously, with the dispatcher updating the status and results services to allow callers to determine job status and retrieve results.

The dispatcher determines whether the given query requires EDC data retrieval by comparing the classes in the incoming nodegroup to the stored EDC configuration, which is contained in its own graph in the semantic triple store. If any field in the incoming nodegroup is a subclass of ExternalDataValue (a class in the EDC semantic model), then EDC will be invoked. If EDC operations are not required, the dispatcher acts as a pass-through to the SPARQL query service, fulfilling the user query using the semantic store only. If EDC operations are required, the dispatcher consults the EDC configuration to determine what metadata is required for the external invocation, and augments the incoming nodegroup with the additional metadata. For example, this augmenting metadata may include the location of the database server containing external data, and the database column names to retrieve. The semantic query results are then binned, with each bin containing results corresponding to a single query against an external store. For example, the bin corresponding to a time series request for a given asset may contain the name, site, and install date of the asset in question. Each semantic result bin is given a unique identifier (UUID) to simplify associating the semantic results with the EDC results. The dispatcher then calls the external query generation and execution services, to respectively generate and execute queries on external data stores.

After the completion of the external query, the dispatcher is responsible for fusing the external results with the results from the semantic portion of the query, thereby returning the external results with the proper context.

b) EDC Query Generator(s)

The EDC query generator is a not a true component, but rather the specification for a component which accepts a given input format and outputs queries to be run on a given external service or data source. The input to the query generator is a table containing, at minimum, a collection of UUIDs of semantic results bins. The input also includes any required metadata, as defined in the EDC configuration. For example, a Hive Query Generator generates a table of queries that can be run by the Hive Query Executor against a Hive installation. Relevant input data for the Hive Query Generator may include table names and column names in Hive.

The EDC query generator output may not be a query in the traditional sense. Rather, it is a structure that encapsulates the information required to interact with an external service. This insulates the dispatcher and other components from knowledge of any external data store, thus enabling external stores to be readily replaced. The query generator is task-specific to both the structure of the external data and the requirements of the external system. The generator returns a table of queries, one per input UUID, which will be used by the query executor.

c) EDC Query Executor

The EDC query executor is also a specification for a component rather than a discrete implementation. Like the query generator, the executor is specific to a given data source and use case. The executor accepts the structure outputted by the matching query generator and uses the contents to invoke the service and return the resulting data. As with the generator, the executor insulates the dispatcher from any understanding of the external service's operations. Any executor accepts the table of inputs authored by the paired generator and maintains the UUID association. When the executor has completed, it returns a table of the results to the dispatcher, appending the correct original UUIDs to each of the returned records.

d) Status Service

The status service acts as an intermediary between the dispatcher and caller, enabling asynchronous services to be used. At a regular interval, the dispatcher writes its current state to the status service, indicating how much progress has been made on the job. The dispatcher determines the progress by partitioning the workload into subtasks with estimated time weights as follows: set up and initial sematic query 30%, external query 50%, and results fusion 20%.

e) Results Service

The results service accepts the fused results from the dispatcher and provisions them to a storage location where they can be retrieved by the original caller. The service uses a dedicated graph in the semantic store to keep track of where the results are provisioned.

3) SemTK User Interface Components

The user interface components allow users unfamiliar with the Semantic Web tech stack to interact with semantically-enabled data. Additionally, the UI components simplify the experience of interacting with the data in the semantic store by exposing functionality that can directly interact with instance data. The filter dialogues are capable of directly querying instance data allowing for filters based on both data already present as well as regular expressions. The UI allows for previews of query responses, saving connection information, and mapping a nodegroup's properties to entries in a data source for use during ingestion. Lastly, the UI allows the saving and restoring of a user's session through the import and export of serialized nodegroups.

D. Adding New External Services

The SemTK EDC supports fast and flexible integration of additional services to meet evolving data access needs. This can be achieved without any code changes to SemTK itself. To make a new EDC data source available to the system, the follow actions must be taken:

1. Extend the EDC semantic model to include a subclass for the return of the new service's data.
2. Implement an EDC query generator for the new data source, using the API definition provided in SemTK's service tier.
3. Implement an EDC query executor for the new data source, which accepts the output of its paired generator and retrieves data from the service.
4. Make the query generator and query executor services available via REST service calls.
5. Add an EDC configuration for the new EDC class, specifying the generator and executor service locations and their respective metadata needs.

Once properly wrapped and added to the registry as outlined above, the new source is immediately available.

IV. Use Case Descriptions

SemTK has enabled several use cases that required access to information in Big Data stores, augmented by context relevant to the data. In these cases, the Big Data was required to be retrieved in an ad-hoc manner, with users and systems defining the queries on demand. Further, it was required that users could search the data without knowledge of the underlying data stores or structures. Using SemTK and EDC for these use cases allowed subject matter experts to search for Big Data and corresponding context in a single logical interface using domain terms.

A. Turbine Engineering Data System

One of the primary use cases that motivated this work was the Turbine Engineering Data System (TEDS) for GE Power [12]. GE Power's engineering division performs large scale testing of turbines and turbine components, producing large quantities of test measurement data (1Hz data collected for 10,000+ parameters for hours to months per test) as well as extensive test configuration data (100's of parameters per test). Overall, a single test will generate gigabytes to many terabytes of raw data.

The previous approach to storing and accessing this Big Data did not allow for easy access. First, the test configuration data was stored separately from the test measurement data, with no codification of how they relate to each other and no capability for integrated querying. Second, the test measurement data is collected from many different sensors and calculations, with significant variation across tests, such that, for example, column DA_348 may contain temperature measurements in one test and pressure measurements in another. This variable mapping information was not stored in either of the above data stores, making this information at times dependent on institutional memory. Thus, performing a query such as "retrieve emissions measurements and combustor temperature for tests run on Test Stand 5 in the last 6 months" required first querying the test configuration store for the relevant test numbers, manually accessing a document to identify the relevant column names, querying the test measurement storage to retrieve those columns for each test, and then collating the results. This data collection process could take days or weeks to complete depending on the complexity of the query, with significant manual effort.

GE Power needed a solution to allow: (i) querying of configuration and measurement data in one logical interface, accessible by both humans searching using domain terms and by analytics programmatically retrieving data, (ii) continued allowance for configuration data and time series measurement data to be stored physically separate in stores best suited to each data type, (iii) deployment in a scalable, robust cloud environment, and (iv) extensible to different kinds of GE Power data (e.g. operational data from the remote monitoring and diagnostics division).

The factors above motivated the initial development of the SemTK EDC service described above. In this instance, the test measurement (time series) data is stored in Apache Hive, whereas the test configuration data is stored directly in the semantic store. The semantic store also contains metadata describing the time series in Hive, and the variable mapping information.

The service-based architecture has enabled TEDS to be deployed in Amazon Web Services, with different services running on different EC2 nodes. Multiple instances of each service can be stood up, supporting scalability and high availability. Further, the capability of executing queries via a REST API means that the data can be programmatically accessed for use in analytics or visualizations. Finally, the ability to configure the system to use varying external query generators and query executors means that TEDS is extensible for use with other types of Big Data at GE Power. Because of SemTK, data retrievals that took days or weeks to complete now take minutes.

B. *Digital Twin – Linked Asset Config and Instance Data*

A second use case that motivated this work was the need for advanced domain-specific query capabilities across a multitude of diverse industrial datasets within GE's Digital Twin Framework. The Digital Twin is a paradigm wherein virtual incarnations ('twins') of physical assets continuously learn based on the operation of the asset to facilitate complex business decisions (e.g. performance optimization, predictive maintenance to minimize downtime).

GE's industrial businesses have remote monitoring and diagnostics (RM&D) divisions that analyze operational data from global assets to detect or predict faults. Sample RM&D requirements involve queries of the form: "retrieve all alarms that were fired for a specific asset; next, retrieve all other assets for which this set of alarms fired, filtered by a specific time range and/or geographic location; finally, retrieve the average experience level of the manager in charge of those sites where those assets are operating."

In particular, some businesses may select to use a property graph model to represent their asset data for the following reasons: (i) property graphs are implemented within generalized graph databases that have support for both OLTP-style traversals (e.g., path-finding) and large-scale OLAP-style graph analytics (e.g., page rank), (ii) property graph databases demonstrate excellent horizontal scaling capabilities, making them well-suited for high-volume, high-velocity connected data, and (iii) property graph models allow entities and relationships to have rich, complex properties that can be indexed and searched upon. However, property graphs also have drawbacks. As the graph schema is not embedded within the graph itself, there is no standard, established way that users and applications can interrogate the model to query data in the graph. A priori knowledge of the graph schema is required before any graph traversals are performed. Next, it is challenging to encapsulate domain knowledge in property graphs. Capabilities such as subclass inheritance, optional traversals of subgraphs and inferencing that come with W3C-compliant semantic triple stores are extremely difficult to engineer into property graphs. Finally, some property graph databases impose restrictions on the number of distinct entity and relationship types that may exist in a single graph, limiting the level of expressivity of the ontology.

Thus, the GE RM&D divisions need a solution that can integrate queries of a semantic knowledge base of asset configuration information with property graph databases containing asset-related instance data (such as alarms and site information) using a common interface. This would allow them to continue to store asset instance data in a property graph database of their choice. For this use case, we adapted our services-based solution to enable querying across both a semantic store and the DataStax Enterprise (DSE) Graph database [13]. The semantic store contains the knowledge base of asset configurations as well as metadata describing the asset instances in DSE Graph and the variable mapping information. Using the SemTK framework, we demonstrated that we can automatically generate parameterized Gremlin traversals at runtime, execute the resulting traversals on a remote instance of DSE Graph, retrieve the partial results and fuse them with information obtained directly from the semantic store.

C. *Digital Twin – Gold Data*

Similar to the TEDS use case described previously, this use case also involved using the semantic store to describe Big Data stored in Hive. The goal of this use case was to allow users to capture and query datasets that are considered gold standards for building models. These "gold datasets" most often consist of time series Big Data, but could also be images, documents, or other formats. Whatever the format, the gold datasets are best captured in the most suitable storage mechanisms. SemTK and EDC allow for these Big Data sets, in whatever appropriate format, to be described using semantic metadata and made available for discovery by modelers. For example, modelers may search on what gold datasets exist for a specific model or type of equipment, the creator of the gold data, data quality, or many other parameters.

D. *File Retrieval from HDFS*

A fourth use case involved retrieving industrial asset image files from HDFS using metadata stored in the semantic store. A large collection of image files, capturing the state of parts during nondestructive test evaluations, was loaded into the Hadoop Distributed File System (HDFS). For each image, metadata was captured in the triple store describing the characteristics of the part and the setup or configuration for the test that had been performed. Finally, metadata for the HDFS service and the locations of the individual images was linked to the descriptive metadata.

When a user needs to search for a set of images, they can use a combination of characteristics of the evaluation setup, results, and/or part being tested, to retrieve links to the location(s) of the images of interest. This represented a significant improvement over the previous process, which required externally-hosted, difficult to search information about the parts, evaluations performed, and the images.

Together, these four use cases demonstrate that SemTK is flexible enough to facilitate access to data in a variety of data stores, across a variety of applications, with non-semantic Big Data residing in varying types of storage including in property graphs, Apache Hive, and HDFS.

V. Related Work

We summarize the state-of-the-art in broad research areas related to integrated access to semantic and Big Data sources, and then discuss how the SemTK framework improves upon those. The ubiquitous nature of Big Data has led the database and broader data management community to evolve from traditional ETL and data warehouse practices to polystores and data virtualization. Ontology-based Data Access (OBDA) represents an analogous, nascent effort within the Semantic Web community.

A. *Federated Querying and Data Virtualization*

Approaches for federated querying across multiple relational database systems with a common schema [14][15] or different schemas [16][17] have been proposed and adopted. While all semantic knowledge bases share a common data model based on W3C standards (RDF/OWL), federated querying across multiple knowledge bases hosted on a single semantic store [18] and across multiple semantic stores [19][20] is possible. In contrast with data warehouses and single semantic stores, SemTK allows federated querying across a semantic store and several non-semantic small and Big Data stores via a single, logical interface.

Data Virtualization (DV) is an associated paradigm where applications access data without knowing how or where it is physically stored, and is often viewed by enterprises as an alternative to ETL. Traditional vendors in this space include Denodo, IBM and Informatica. Datometry's Hyper-Q [21] provides adaptive data virtualization for financial services whereby applications can run against different database systems (Postgres, kdb) unmodified. SemTK also seeks to virtualize semantic and non-semantic data sources but using a knowledge-driven interface.

B. *Polystore Systems*

Open-source distributed query engines like Spark SQL [22] and Presto [23] allow interactive querying and analytics against a variety of data sources in a polystore setting, but put the onus of cross-store querying on the user. Some integration approaches [24][25] tightly integrate the Hadoop Distributed File System (HDFS) with RDBMSs but do not consider other data models. Similarly, approaches such as [26][27] allow integrated querying across multiple NoSQL stores alone. SQL++ [28] and Apache Drill [29] allow for interactive querying across RDBMSs and potentially multiple schema-less data sources.

Recently, more generic and extensible polystore systems are being developed to accommodate federated querying across a larger number of diverse data stores. Examples in this category include BigDAWG [30], PolyFuse [31], Proteus [32], Estocada [33], CloudMdsQL [34] (part of the CoherentPaaS platform) and QUEPA [35]. Users of these systems either query the polystore using the query language and data model with which they are most comfortable, or via some higher-level language. These systems transform queries into subqueries that are each executed by their respective storage engines, and fuse the results. Digree [36] and MELOGRAPH [37] are polystores that exclusively focus on integrating across multiple graph stores. SemTK allows integrated access to both semantic stores and traditional polystores in the context of a domain model.

C. *Ontology-based Data Access*

Ontology-based Data Access (OBDA) technology provides mechanisms for the retrieval of data from polystores using information in a semantic layer. An ontological description of a domain is used to generate domain-specific queries that are then satisfied by using a translation layer to map external data sources to the model terms. Rather than store the actual data in a triple store, information about how to map the terms in a domain model to data in the underlying storage substrate is captured. Through this mapping, semantic queries are translated into a set of subqueries each executed by the underlying data stores. The results are combined and returned to the caller. OBDA systems such as Capsenta's Ultrawrap [38] rely on general-purpose R2RML [39] translations between a relational data storage layer and a semantic domain model to enable retrieval from several databases. Solutions such as MASTRO [40] use custom, purpose-built pipelines to manage data sources, write queries and return data.

Unlike OBDA techniques which store the mapping separately, we store knowledge that is likely to be searched based on the domain directly in the triple store, allowing for graph queries to be formulated directly. The large (or otherwise unsuitable) data is stored remotely in the polystore, allowing a large amount of data to be controlled and accessed by a relatively small configuration. Also, unlike traditional OBDA solutions, we do not assume that all external data mapped to a single property must come from a single underlying store.

D. *Interfaces for Knowledge-driven Access to Data*

A particularly unique aspect of SemTK is the use of knowledge in the knowledge graph to drive the search for and access to disparate datasets. While [41] focuses on query rewriting for multiple RDBMSs, we focus on services that auto-generate simple yet parameterizable queries for many external data stores including Hive and DSE Graph. QUEPA's auto-exploration mode [35] allows users to expand on query results and incrementally build more complex queries with the ability to go back to a previous query result. SemTK's nodegroups allow similar incremental exploration capability. OptiqueVQS [42] is an ontology-based visual query formulation system that enables auto-generation of SPARQL queries based on input concepts and constraints provided by users via a graphical interface. Like SemTK, it allows users to formulate their queries using terms in their domain.

E. *Discussion of Contributions*

While we share high-level motivations with traditional polystore systems like BigDAWG [30], our work differs in several crucial ways: (i) SemTK allows users to query the data using domain knowledge terms, whereas polystores require users to be familiar with at least one data model and language (or meta-model and meta-language). (ii) SemTK allows integrated querying across a semantic store and non-semantic Big Data stores. (iii) Polystore systems focus on optimizing queries spanning multiple storage engines and on automated load balancing. While we intend to address these issues in the future, our work currently focuses on developing knowledge-driven interfaces to polystores.

To our knowledge, our SemTK framework and the Optique system [43] being developed in parallel, are the first attempts to introduce a knowledge-driven approach for integrated access to semantic data as well as non-semantic Big Data polystores. Whereas Optique appears to focus on reconciling specific data types (e.g. static, streaming, geospatial) using corresponding custom mapping languages (e.g. STARQL, GeoSPARQL), ours is a generic plug-and-play services-oriented framework to support external data retrieval from a wide variety of traditional data sources described and linked by a common ontology. Optique provides powerful ontology-driven query capabilities within supported types; SemTK, through an emphasis on modular design, provides easier capabilities to add support for new data stores and formats as needed. Another point of differentiation is that Optique uses ontologies and semantics solely for mapping to external stores, whereas SemTK allows (and its use cases call for) portions of the data to be stored directly within the semantic triple store.

VI. Conclusions and Future Work

In this paper we describe SemTK, a framework that leverages semantic technology and extends traditional knowledge graph models to provide knowledge-driven integrated access to data stored across both semantic *and* non-semantic Big Data polystores. This allows all descriptive metadata (e.g. configuration information) that are useful for search and indexing to be maintained in a semantic store, whereas any other form of data not well-suited for native storage in a semantic layer can be persisted in other data stores that are external to the semantic store. Thus, high volume/velocity data with low knowledge content can continue to reside in data stores most appropriate to their respective data types (e.g., time series data in historians or time-series databases). SemTK provides a mechanism that enables such external data sources in a polyglot-persistent setting to be linked in accordance with a shared model of the user's domain, such that the data can be retrieved seamlessly and transparently.

A key benefit of adopting the SemTK framework is that domain knowledge becomes a transferable asset. In typical enterprise Big Data environments, users are required to have knowledge of IT system details to retrieve the data they need, thus severely restricting the pool of potential users. SemTK represents a move towards IT system independence, allowing the caller to request information in domain terms they understand. It does not require that the data be stored in any particular storage mechanism or that users be aware of how to execute queries on it. In contrast, any user familiar with a given domain can query for information regardless of the underlying data systems, provided a domain model and appropriate external data connectors exist. Any time new (potentially Big Data) data sources are introduced, SemTK's modular services-based architecture makes it easy to develop and deploy connectors to those external sources.

The power of the SemTK framework has been proven through the significant value it has brought to GE, accelerating the development of knowledge-driven applications by simplifying the integration and use of data found across diverse data stores. We demonstrate the utility of SemTK in the industrial domain via four use cases ranging from turbine engineering to remote monitoring and diagnostics. An open source version of the core SemTK system is available at https://github.com/ge-semtk/semtk.

There are several directions for future research and development of SemTK. We intend to add the capability to integrate other kinds of storage paradigms (such as NoSQL databases and array databases) that may be of interest to the industrial domain. In addition to static data sources, SemTK's EDC configuration could also be expanded such that external data services can calculate or sample data from dynamic, even real-time, sources including persistent message queues. In this way, the framework could allow for the execution of external services or applications, such as Spark or Hadoop MapReduce jobs to fulfill user requests, beyond purely querying data stores. While SemTK can fuse results from multiple stores to serve a single request, we currently do not perform any cross-store queries. Support for joins and other re-entrant query types across stores presents another challenging research problem. We also intend to explore caching of partially or completely fused results in SemTK such that future nodegroup requests can be served from a cache based on similarity to prior nodegroups. Finally, the user interface could be improved to find a better balance between complexity that is exposed to the user and ease of use, for example, striking a balance between automatic pathfinding and query flexibility.

ACKNOWLEDGMENTS

The authors would like to thank Steven Gustafson, David Hack, Matt Nielsen, Ryan Oattes, and Eric Pool.

REFERENCES

[1] Google – Inside Search. The Knowledge Graph, 2017. https://www.google.com/intl/bn/insidesearch/features/search/knowledge.html

[2] Bing blogs – Understanding your World with Bing, 2013. http://blogs.bing.com/search/2013/03/21/understand-your-world-with-bing/

[3] Q. He. "Building the LinkedIn Knowledge Graph", 2016. https://engineering.linkedin.com/blog/2016/10/building-the-linkedin-knowledge-graph

[4] Facebook Engineering – Under the Hood: Entities Graph, 2013. https://www.facebook.com/notes/facebook-engineering/under-the-hood-the-entities-graph/10151490531588920/

[5] M. Stonebraker, "The Case for Polystores", http://wp.sigmod.org/?p=1629, Jul 2015, accessed 2017.

[6] D. Vrandecic, M. Krotzsch. "Wikidata: A Free Collaborative Knowledge Base", in *Comm. of the ACM*. vol. 57, pp. 78-85, 2014.

[7] J. Lehmann et al. "DBpedia – A Large-scale, Multilingual Knowledge Base Extracted from Wikipedia". *Semantic Web Journal*. vol. 6, no. 2, pp. 167-195, 2015.

[8] F. Mahdisoltani, J. Biega, F. M. Suchanek. "YAGO3: A Knowledge Base from Multilingual Wikipedias", in *Proc. of 7th Conf. on Innovative Data Systems Research (CIDR)*, Asilomar, CA, 2015.

[9] R. Blanco, B. B. Cambazoglu, P. Mika, N. Torzec. "Entity Recommendations in Web Search", in *Proc. of 12th Intl. Semantic Web Conf. (ISWC)*, Sydney, Oct 2013, pp. 33-48.

[10] L. Ehrlinger, W. Wöβ. "Towards a Definition of Knowledge Graphs". SEMANTiCS 2016.

[11] P. Cuddihy, J. McHugh, J. Weisenberg Williams, V. Mulwad, K.S. Aggour. "SemTK: An Ontology-first, Open Source Semantic Toolkit for Managing and Querying Knowledge Graphs", *arXiv:1710.11531 [cs.AI]*, 2017.

[12] J. Weisenberg Williams, P. Cuddihy, J. McHugh, K.S. Aggour, "Semantics for Big Data Access & Integration: Improving Industrial Equipment Design through Increased Data Usability", in *Proc. of the IEEE International Conference on Big Data*, pp. 1103-1112, 2015.

[13] DataStax Enterprise Graph, http://www.datastax.com

[14] M. Stonebraker *et al.* "Mariposa: A Wide-area Distributed Database System", in *The VLDB Journal*, vol. 5, pp 48-63, Springer, 1996.

[15] M. J. Carey *et al.* "Towards Heterogeneous Multimedia Information Systems", in *Data Engineering: Distributed Object Management*, pp 124-131, IEEE, 1995.

[16] R. Pottinger and P. A. Bernstein. "Schema Merging and Mapping Creation for Relational Sources", in *Proc. of EDBT*, pp 73-84, 2008.

[17] C. Batini, M. Lenzerini, and S. B. Navathe. "A Comparative Analysis of Methodologies for Database Schema Integration", in *ACM Comp. Surveys*, vol. 18(4), pp. 323-364, 1986.

[18] Carlos Buil-Aranda. "Federated Query Processing for the Semantic Web", Studies on the Semantic Web, vol. 15, IOS Press, Jan 2014.

[19] Revelytix releases Spyder 1.0, http://www.businesswire.com/news/home/20120209006475/en/Revelytix-Releases-Spyder-1.0, 2017.

[20] M. Schmidt, O. Gorlitz, P. Haase, G. Ladwig, A. Schwarte, T. Tran. "FedBench: A Benchmark Suite for Federated Semantic Data Query Processing", in *Proc. of Intl. Semantic Web Conf. (ISWC)*, pp. 585-600, 2011.

[21] L. Antova *et al.* "Datometry Hyper-Q: Bridging the Gap Between Real-Time and Historical Analysis", in *Proc. of ACM SIGMOD*, pp. 1405-1416, 2016, San Francisco.

[22] M. Armbrust *et al.* "Spark SQL: Relational Data Processing in Spark", in *Proc. of ACM SIGMOD*, pp. 1383-1394, 2015.

[23] M. Traverso. "Presto: Interacting with Petabytes of Data at Facebook". Blog post. https://research.fb.com/presto-interacting-with-petabytes-of-data-at-facebook/, Aug 2014, accessed 2017.

[24] A. Abouzeid, K. Badja-Pawlikowski, D. Abadi, A. Silberschatz, A. Rasin, "HadoopDB: An Architectural Hybrid of MapReduce and DBMS Technologies for Analytical Workloads", in *Proc. of VLDB*, pp. 922-933, Aug 2009, Lyon, France.

[25] J. LeFevre, J. Sankaranarayanan, H. Hacigumus, J. Tatemura, N. Polyzotis, M. J. Carey, "MISO: Souping up Big Data Query Processing with a Multistore System", in *Proc of ACM SIGMOD*, pp. 1591-1602, Jun 2014, Snowbird, Utah.

[26] UnQL: A Standardized Query Language for NoSQL Databases, http://www.dataversity.net/unql-a-standardized-query-language-for-nosql-databases/, Mar 2012, accessed 2017.

[27] ArangoDB: "One Core. One Query Language. Multiple Data Models", https://www.arangodb.com/, accessed 2017.

[28] K. W. Ong, Y. Papakonstantinou, R. Vernoux, "The SQL++ Semi-structured Data Model and Query Language", in *CoRR*, vol. abs/1405.3631, 2014.

[29] M. Hausenblas and J. Nadeau, "Apache Drill: Interactive Ad-Hoc Analysis at Scale", in *Big Data*, vol. 1(2), pp. 100-4. Jun 2013.

[30] J. Duggan *et al.* "The BigDAWG Polystore System", in *ACM SIGMOD Record*, vol. 44(2), pp. 11-16, Jun 2015.

[31] M. Gubanov, "PolyFuse: A Large-scale Hybrid Data Integration System", *Proc. of IEEE ICDE DESWeb*, 2017, San Diego.

[32] M. Karpathiotakis, I. Alagiannis, A. Ailamaki, "Fast Queries over Heterogeneous Data through Engine Customization", in *Proc. of VLDB Endowment*, vol. 9(12), pp. 972-983, Aug 2016.

[33] F. Bugiotti, D. Bursztyn, A. Deutsch, I. Ileana, I. Manolescu, "Invisible Glue: Scalable Self-Tuning Multi-Stores", in *Proc. of CIDR*, Jan 2015, Asilomar, CA.

[34] B. Kolev, P. Valduirez, C. Bondiombouy, R. Jimenez-Peris, R. Pau, J. Pereira, "CloudMdsQL: Querying Heterogeneous Cloud Data Stores with a Common Language", in *Distributed and Parallel Databases*, vol. 34(4), pp. 463-503, Dec 2016.

[35] A. Maccioni, E. Basili, R. Torlone, "QUEPA: Querying and Exploring a Polystore by Augmentation", in *Proc. of ACM SIGMOD*, pp. 2133-2136, 2016, San Francisco.

[36] V. Spyropoulos, C. Vasilakopoulou, Y. Kotidis, "Digree: A Middleware for a Graph Databases Polystore", in *Proc. of IEEE Conf. on Big Data*, 2016, Washington D.C.

[37] C. E. Ciolac, "MELOGRAPH: Multi-Engine WorkfLOw Graph Processing", in *Proc. of Workshops of the EDBT/ICDT joint Conf.*, 2016.

[38] J. Sequeda and D. Miranker. Ultrawrap: SPARQL Execution on Relational Data. Web Semantics: Science, Services and Agents on the World Wide Web, North America, 22, 2013.

[39] R2RML: RDB to RDF Mapping, https://www.w3.org/TR/r2rml/

[40] D. Calvanese *et al.* "The MASTRO System for Ontology-based Data Access", The Semantic Web Journal, 2(1):43-53, 2011.

[41] N. Abolhassani, T. Tung, K. Gomadam, L. Ramaswamy, "Knowledge Graph-based Query Rewriting in a Relational Data Harmonization Framework", in *Proc. of Intl. Conf. on Collaboration and Internet Comp.*, Nov 2016, Pittsburgh.

[42] A. Soylu, M. Giese, E. Jimenez-Ruiz, G Vega-Gorgojo, I. Horrocks, "Experiencing OptiqueVQS: A Multi-Paradigm and Ontology-based Visual Query System for End Users", in *Univ. Access in the Info. Society.*, vol. 15(1), pp. 129-152, Mar 2016.

[43] E. Kharmalov *et al.* "A Semantic Approach to Polystores", in *Proc. of IEEE Intl. Conf. on Big Data*, 2016, Washington D.C.