Helix: Online Enterprise Data Analytics

Oktie Hassanzadeh[1,2], Songyun Duan[1], Achille Fokoue[1],
Anastasios Kementsietsidis[1], Kavitha Srinivas[1], Michael J. Ward[1]
[1]IBM T.J. Watson Research Center
Hawthorne, NY, U.S.A.
[2]University of Toronto
Toronto, Ontario, Canada

ABSTRACT

The size, heterogeneity and dynamicity of data within an enterprise makes indexing, integration and analysis of the data increasingly difficult tasks. On the other hand, there has been a massive increase in the amount of high-quality open data available on the Web that could provide invaluable insights to data analysts and business intelligence specialists within the enterprise. The goal of Helix project is to provide users within the enterprise with a platform that allows them to perform online analysis of almost any type and amount of internal data using the power of external knowledge bases available on the Web. Such a platform requires a novel, data-format agnostic indexing mechanism, and light-weight data linking techniques that could link semantically related records across internal and external data sources of various characteristics. We present the initial architecture of our system and discuss several research challenges involved in building such a system.

Categories and Subject Descriptors

H.3.5 [**Information Storage and Retrieval**]: Online Information Services; H.2.4 [**Database Management**]: Systems

General Terms

Algorithms, Design

Keywords

Enterprise Data Management, Data Integration, Linked Data

1. INTRODUCTION

Businesses ranging from small companies to large corporations are facing today a *data explosion* [1, 5, 6]: every day, businesses accumulate massive amounts of data from a variety of sources, and they employ an increasing number (often measured in the thousands) of heterogeneous, distributed, and often legacy data repositories to store them. Businesses are already spending considerable effort to manage their data, by having in place data warehousing, data integration, or Extract-Transform-Load (ETL) solutions. Part of their challenge is to extend these existing solutions to deal with the *scale* of this data explosion, to meet the continuing requirements of their day-to-day operations. But an increasingly important challenge for most businesses facing this data explosion is one of *semantics*: *Businesses don't know what they don't know.*

WWW 2011, March 28–April 1, 2011, Hyderabad, India.
ACM 978-1-4503-0637-9/11/03.

Indeed, while existing data analytics solutions do a great job in collecting data and providing answers on known questions, key business insights not only remain hidden in the rest of the data, but due to the data explosion they are even more difficult to find. Keyword search is the most popular way of finding information on the Web, and the techniques developed there have a proven track record for dealing with massive amounts of data. However, keyword search is not particularly compelling in the business context considered here. Consider for example a business analyst of a technology company who is interested in analyzing his company's records for customers in the healthcare industry. Given keyword search functionality, the analyst might issue a *"healthcare customers"* query over a large number of repositories. Such a search might identify records like *Clinovia Healthcare* that either mention the keyword *"healthcare"* explicitly, or records like *Cromwell Hospital* that mention some derivative (e.g., synonym, hyponym/hypernum) of the keyword. The search will probably not return a company like *British United Provident Association (BUPA)* in its results even though *BUPA* is a company in the healthcare industry, since no terms in the company's name relate to the keyword query. Even worse, the search will probably return scores of business records from the various repositories with no apparent connection between them. So, it will fail to provide a connection between *BUPA* and *Cromwell Hospital*, since the former has acquired the latter (source Wikipedia), and the fact that *BUPA* is also the parent company of *Clinovia Healthcare* (source Wikipedia).

Our first objective is to develop techniques to not only find, but also *link* information across enterprise repositories; these techniques must be both precise with respect to the user's intent and scalable. Linking between entities across repositories has been the focus in a large number of works (including our latest paper [4]). However, the majority of existing works perform entity linking in a *batch*, *off-line* fashion. With huge data sets, and large number of repositories, these methods will generate every possible link, between all possible linkable entities. Generating thousands of links not only requires substantial computation time and considerable space to store them, but also requires substantial effort since the generated links must be verified and cleaned, given the inherently imprecise nature of linking methods. To compound the problem, the same two entities may be linked for a variety of reasons, each orthogonal to each other. For example, two companies may be linked because one is a subsidiary of the other (as is the case with *BUPA* and *Clinovia*), or they may have the same CEO (e.g., as was the case for

Apple Inc. and *Pixar*). Generating all possible links, along with evidence for the *provenance* of why each link was generated, is clearly impractical. Even if such an approach were feasible, it will be a huge waste of resources to generate and store all these links only to discover later that only a small fraction of them is actually relevant to business analysis.

We address the shortcomings of existing linking approaches by providing a *dynamic* and *context-dependent* linking mechanism. In more detail, we take advantage of user profile metadata, in conjunction with metadata associated with the input keywords (see details in later sections) to link dynamically, i.e., *at query time*, only the entities that reside in different repositories and are potentially relevant to the current search. So in our previous example, our system will try to identify only the healthcare-related companies from the various repositories and to establish links only between those (instead of between all the companies, and instead of establishing arbitrary links as in the case of common CEO's).

The second objective of our work is to provide an extensible framework in which new data sources, both internal and external, can be easily incorporated to address changing business requirements. Enterprises continually produce valuable data using informal methods like spreadsheets and business mashups. These data sources may be critical to evolving analytical tasks. Additionally, data that are useful to business analytics will reside in external sources, no matter how much data an enterprise manages internally. These external sources are often authorities in their fields; trying to duplicate and independently maintain their data within an enterprise makes little sense. Instead, the enterprise should be able to leverage these external sources in its own analytics. In our example, our enterprise might have detailed information for its customers and their transactions, but it is missing data on company acquisitions. These data are stored in external sources, like Wikipedia, Bloomberg or the FCC, or most recently in RSS and Twitter feeds. By leveraging these external sources in our simple example, we are not only able to identify *BUPA* as a company that is in the health-care industry, but also establish links between *BUPA* and its subsidiaries. Indeed, one of the main contributions of this work is blurring the distinction between internal and external sources in the enterprise and offer a framework to seamlessly use both types.

In this demonstration, we will use our existing prototype system and will walk users through the steps of several business analysis tasks. During the presentation, we will issue several live queries and show our system's ability to (a) find records relevant to the input query; (b) *dynamically* (and at *query time*) establish links between the retrieved records; (c) take advantage of external online sources in order to achieve both steps (a) and (b). In the following sections, we present the current architecture of our system, and provide details about the various implemented features. Then, we provide more details about the types of scenarios we are interested in showcasing, and the types of sources we are planning to use for these scenarios.

2. HELIX FRAMEWORK

Figure 1 shows the implemented architecture of the Helix system. In what follows, we present an overview of different components in this framework.

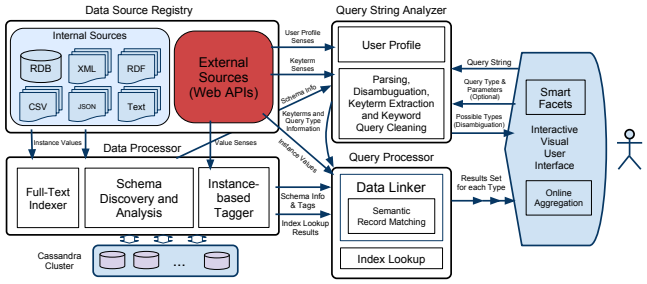

Figure 1: Helix Framework.

2.1 Data Source Registry

One of the primary objectives of the Helix system is to allow analysis of highly heterogeneous enterprise data repositories. Such repositories contain data of different formats (e.g., text/CSV, relational databases, XML, RDF) and varying characteristics (e.g., very large number of small data records, or small number of very large data records). In addition, the system takes advantage of various online data sources, with APIs that support different query languages (e.g., SPARQL over RDF stores, MQL over Freebase API, or keyword search query interface). Our system provides a common interface to all the accessible data sources through the data source registry component. The registry keeps a catalog of available internal and external sources and their access methods and parameters, such as the host name (server IP or Web API URL), driver module (if any), authentication information, and indexing parameters. Users can add data sources to the registry and delete them from it as needed.

2.2 Data Processor

Given a set of available internal and external sources in the registry, the data processor provides the other components with a common access mechanism for all the data. For internal data sources, it provides a level of indexing and analysis depending on the type of data source. Note that no indexing or caching is performed over external sources, i.e., we retrieve fresh data from external sources as needed. For internal sources, the first step in processing is to identify and store schema information and possibly perform data format transformation. Our framework supports legacy data with no given or well-defined schema, as well as semistructured or *schema-free* data. Therefore, apart from looking up the available schema information (for example from an RDBMS data dictionary), we need to perform *schema discovery*, also referred to as *schema extraction*, for some of the sources. We use and extend existing schema discovery techniques [7]. Furthermore, we use a novel tagging mechanism to identify extended data type and semantic information about our data, which we call the *senses* of the data. So, in the case of relational data, our method picks a sample of instance values for each column and issues them as queries to online sources to gather possible senses of the instance values for the column. The result is a set of tags associated with each column, along with a confidence value for the tag. To illustrate, assume a column from a relational database whose values correspond to company names. By polling Freebase, we get a set of sense tags, one set per company name. We correlate the tags from all the values, and at the end of the process annotate the column with senses like *Company*, *In-*

dustry, and possibly *Healthcare* (if most sampled companies are from the healthcare industry). Notice that this process is independent of the relational column name. So, it works even if the column name is some acronym like *COMP_NM*, or some string representing an internal code.

Finally, our system uses a highly efficient full-text index across all internal repositories, powered by a Cassandra[1] cluster. We use different indexing strategies depending on the source characteristics. For a relational source, for example, depending on the data characteristics and value distributions, the indexing is performed over rows (where we index values and store the primary keys of the tuples they appear in), or columns (where we index values and store the column(s) of the relations they appear in). For string values, we build q-gram-based indices to allow fuzzy string matching queries [2]. To identify indexed values, we create URI's that uniquely identify the location of the values across all enterprise repositories. For example, indexing the string *BUPA*, appearing as a value in column *NAME* of a tuple with primary key *CID:34234* in table *CUST*, of source *SOFT_ORDERS*, results in the URI `/SOFT_ORDERS/CUST/NAME/PK=CID:34324`, which uniquely identifies the source, table, tuple and column in which the value appears.

2.3 Query Analyzer

The query analyzer is responsible for processing the input search request and (a) determining the *query type*; and (b) identifying key terms associated with the input query. In more detail, the Helix query interface supports several types of queries, ranging from basic keyword-based index lookup to a range of advanced search options. Users can either specify the query type within their queries or use an advanced search interface. The main function of this component is key term extraction and disambiguation. We build upon previous work on keyword query cleaning [8] to detect possible syntactic errors and semantic differences between the user's query and the indexed data instances as well as to perform segmentation (identification of key terms).

As in many information retrieval systems today, terms in the query string can be modifiers that specify the type or provide additional information about the following term. To permit individual customization, the query analyzer can employ a *user profile* that contains information about user's domain of interest in the form of a set of *senses* derived from external sources. The user profile can be built automatically based on query history or can be built manually by a user. In our motivating example, the user profile for an analyst can include the simple senses *"business"* and *"computer"* to indicate that the analyst is only interested in this particular context. At query time, the senses of the identified input query key terms are found by querying external sources, and these senses are matched with the user profile to assist detection of the type of the key term (i.e., whether it is a modifier or not) and therefore the query type. In our example, if the analyst issues the simple keyword query *"Apple"*, the senses of this term will be retrieved from external sources, including senses like *"Fruit"*, *"Computer company"* and *"Music"* (due to Apple Corp, a corporation founded by The Beatles). These senses will be matched with the user profile, and therefore only the senses related to *"Apple"* as a computer company will be used in any further processing.

[1]The Apache Cassandra Project - `http://cassandra.apache.org/`

2.4 Query Processor

The query processor relies on the information it gets about the input query from the query analyzer in order to process the query and return its results. A key part of the query processor is the data linking module. The query processor might need to issue queries to the internal index as well as online APIs, and put together and analyze a possibly large and heterogeneous sets of results retrieved from several sources. In addition to retrieving data related to the user's queries, the processor may issue more queries to online sources to gain more information about unknown data instances. The data linking module consists of state-of-the-art record matching and linking techniques that can match records with both syntactic and semantic differences [4]. The matching is performed between instances of attributes across the internal and external sources.

To increase both the efficiency and accuracy of the matchings, the attribute tags (senses) created during preprocessing are used to pick only those attributes from the sources that contain data instances relevant to the target attribute values. Once matching of internal and external data is performed, unsupervised clustering algorithms are used for grouping of related or duplicate values [3]. The clustering takes into account the evidence from matching with the external data, which can be seen as performing online grouping of the internal data, as opposed to off-line grouping and de-duplication. This allows us to enhance the quality of the groupings and decrease the amount of preprocessing by avoiding offline ad-hoc grouping of all internal data values.

To illustrate the functionality of the query analyzer and processor, consider an analyst issuing a query string *"healthcare in CUST_INFO"*, in an attempt to analyze internal data about companies in the *healthcare* industry. Given this query, the query analyzer identifies key terms *healthcare* and *CUST_INFO*, and also detects that *healthcare* is an industry, and *CUST_INFO* is a data source name in the registry. Therefore, the analyzer sends two queries to the processor: 1) an index lookup for the whole query string 2) a domain-specific and category-specific query *"industry:healthcare datasource:CUST_INFO"*. The second query results in more complex processing in the query processor. For this query, the query processor issues a query to Freebase API to retrieve all the objects associated with object */en/healthcare* (of type */business/industry*), which includes, among other things, all the healthcare-related companies in Freebase. The data linking module then performs efficient fuzzy record matching between the records retrieved from Freebase and internal data from data source *CUST_INFO*. For effectiveness, only those internal records are retrieved whose associated schema element is tagged with a proper sense such as `/freebase/business/business_operation` that is also shared with the senses of the objects retrieved from Freebase (mostly companies in this example).

2.5 User Interface

The main starting point for the users of our system is a keyword query interface (a screen shot is shown in Figure 2). Our system provides a web interface connected to our Java implementation that runs in the web container of an Apache Geronimo[2] Tomcat open source application server. The user interface interacts with the query analyzer module to guide

[2]Apache Geronimo - `http://geronimo.apache.org/`

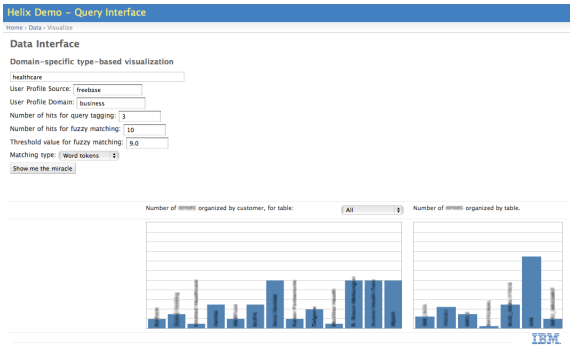

Figure 2: The Helix query web interface.

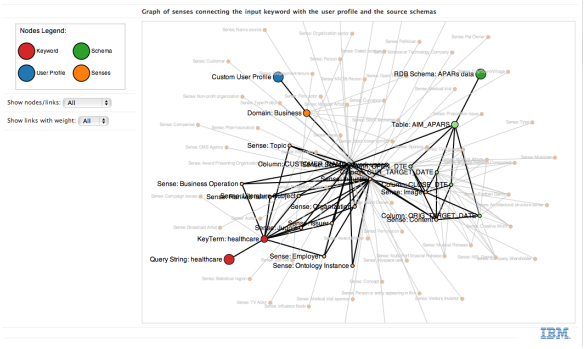

Figure 3: The Helix senses graph.

the user in formulating and fixing the query string. The interface also includes several advanced search features that allow direct specification of query parameters and manually building a user profile. In most cases, more than one query type or set of key terms are identified by the query analyzer. The query analyzer returns a ranked list of possible interpretations of the user's query string, and the user interface returns the top k interpretations along with (a subset of) the results. The users can then modify their query string or pick one query type and see the extended results.

The user interface provides online dynamic aggregation and visualization of the query results. For the graphical rendering of the visualizations of our results (i.e., for the bar-charts, the graphs) we use the Protovis graphical visualization toolkit[3], and the communication between our user interface and the other framework components is done through JSON[4] objects. The interface allows users to pick from multiple ways of aggregating the results for different attributes and data types, along with several types of graphs and charts. At the bottom of Figure 2, we see examples of two such aggregations, represented as bar-charts. Also, the interface provides a glimpse into the system internals, and offers a visual representation of the internal structures and of the reasoning used to identify and return the presented results. An example of such visualization is shown in Figure 3 where the senses associated with the input query, the user profile, and the underlying enterprise data are plotted and correlated so as to illustrate the key senses used in the retrieval and matching of instance data.

3. THE DEMONSTRATION

In this demonstration, we will walk users through the steps of several data analysis tasks like the one described in previous section. In developing and testing Helix, we use real data from IBM's internal customer relationship management (CRM) repositories including customer service and technical support logs. However, we cannot demonstrate Helix using this data directly, for confidentiality reasons. Instead, we will generate synthetic datasets using the schemas and attributes of our internal sources. For attributes like company names or product names, we will randomly pick values of the same category from Wikipedia, maintaining the number of unique entries and their distribution from the original datasets. To expose the data quality issues such as misspellings and alternative representations that are found in the original data, we will introduce a very small amount of syntactic errors (by character deletion, replacement and swapping) or replacements (such as replacing "Inc." with "Incorporated") in the values picked from Wikipedia. For attributes like dates or coded fields, we will randomly generate values in the same ranges found in the original datasets. In this way, we will be able to fully show all the features of our implemented prototype and how the system performs online analysis of enterprise data. To further show that the techniques we employ in our system are not domain-dependent, we also perform online analysis of two real data sources consisting of IMDb movie data, and clinical trials data from ClinicalTrials.gov, stored and treated as internal data.

[3]Protovis - `http://vis.stanford.edu/protovis/`
[4]JavaScript Object Notation - `http://www.json.org/`

4. REFERENCES

[1] B. Babineau. *IBM Information Infrastructure Initiative Tames the Information Explosion*. IBM White Paper, April 2009.

[2] A. Chandel, O. Hassanzadeh, N. Koudas, M. Sadoghi, and D. Srivastava. Benchmarking Declarative Approximate Selection Predicates. In *ACM SIGMOD Int'l Conf. on the Mgmt. of Data*, pages 353–364, 2007.

[3] O. Hassanzadeh, F. Chiang, R. J. Miller, and H. C. Lee. Framework for Evaluating Clustering Algorithms in Duplicate Detection. *Proceedings of the VLDB Endowment (PVLDB)*, 2(1):1282–1293, 2009.

[4] O. Hassanzadeh, A. Kementsietsidis, L. Lim, R. J. Miller, and M. Wang. A Framework for Semantic Link Discovery over Relational Data. In *Proc. of the ACM Conf. on Information and Knowledge Management (CIKM)*, pages 1027–1036, 2009.

[5] E. Knorr. Enterprise Data Explosion Will Only Get Bigger. `http://computerworld.co.nz/news.nsf/management/enterprise-data-explosion-will-only-get-bigger`, August 2010. [Online; accessed 14-October-2010].

[6] S. Lohr. A Data Explosion Remakes Retailing. `http://www.nytimes.com/2010/01/03/business/03unboxed.html`, January 2010. [Online; accessed 14-October-2010].

[7] R. J. Miller and P. Andritsos. On Schema Discovery. *IEEE Data Engineering Bulletin*, 26(3):40–45, 2003.

[8] K. Q. Pu and X. Yu. Keyword Query Cleaning. *Proceedings of the VLDB Endowment (PVLDB)*, 1(1):909–920, 2008.