D-Hive: Data Bees Pollinating RDF, Text, and Time

Srikanta Bedathur[†], Klaus Berberich[‡], Ioannis Patlakas[‡],
Peter Triantafillou[*], Gerhard Weikum[‡]

[‡]Max-Planck Institute for Informatics, Saarbrücken, Germany
[†]IIIT-Delhi, New Delhi, India
[*]University of Patras, Patras, Greece

{kberberi, patlakas, weikum}@mpi-inf.mpg.de, bedathur@iiitd.ac.in, peter@ceid.upatras.gr

ABSTRACT

Although the problem of integrating IR and DB solutions is considered "old", the increasing importance of big data analytics and its formidable demands for both enriched functionality and scalable performance creates the need to revisit the problem itself and to see possible solutions from a new perspective. Our goal is to develop a system that will make large corpora aware of entities and relationships (ER), addressing the challenges in searching and analyzing ER patterns in web data and social media. We put forward D-Hive, a system facilitating analytics over RDF-style (SPO) triples augmented with text and (validity / transaction) time capable of addressing the functionality and scalability requirements which current solutions cannot meet. We consider various alternatives for the data modeling, storage, indexing, and query processing engines of D-Hive paying attention to the challenges that must be met, which include i) scalable joint indexing of SPO-text-time tuples (quads, quints, octs, etc.), ii) efficient processing of complex queries that involve RDF star and path joins, filtering and grouping on text phrases, band joins over time, and more, as well as iii) optimizing the execution plans for such analytics.

1. MOTIVATION

DB&IR integration has been a research topic for the last decade, but the results have made little impact. In the era of big-data analytics, new opportunities are arising for two reasons: i) there is a wealth of data on the Web and in social media that can be made fully coherent only by querying and analyzing both structured data and textual phrases and by placing them in the right temporal context, and ii) there is a sea change in industry in trying and adopting new kinds of NoSQL platforms and non-traditional data engines.

Novel data syntheses at scale emerge in several ways:

- Advances on information extraction (IE) and the Linked Open Data (LOD) initiative have enabled the construction of huge *knowledge bases* like Yago, DBpedia, or Freebase (the latter powering the Google Knowledge Graph). These are extensively interconnected through entity-level sameAs links with each other and with further external references to data on music, books, and much more [3, 9]. In total, the Web of *Linked Data* consists of more than 30 Billion SPO triples and keeps growing.
- Using IE methods and techniques from data integration, it has become possible to interpret *Web tables* and lists in HTML (or Excel etc.) in a semantically informative manner [14], reconstructing column domains and mapping cell values onto entities. Google Fusion Tables, arguably the largest project of this kind, contains tens of millions of tables and keeps growing.
- Structured annotations can be embedded in HTML pages using RDFa or *microformats*, and are gaining momentum. The industry standard *schema.org* escape the dilemma of name heterogenity and the need for semantic matching by defining a common vocabulary for types, attributes, and relationships on a large and rapidly growing number of entity classes (e.g., Brewery, MotorcycleDealer, InfectiousDisease, Campground, etc.). *sindice.com* has indexed 600 million pages containing many billions of SPO triples.

Our thesis is that the structured data/knowledge is only a partial view of the full contents, derived from Web pages, news articles, or social-media postings. For insightful analytics and coherence it is necessary to combine the structured parts with text predicates, e.g., to tap into entity names (people, places, products), marketing slogans, quotations, user comments, etc. Moreover, all this needs to be properly positioned in time; the temporal dimension is crucial for making sense of data from different periods (e.g. in news archives or Internet archives). For example, when combining knowledge-base facts on Angela Merkel with her Wikipedia article, we can find her as an answer to a query for female politicians from Germany (expressed in the RDF query language Sparql) who worked on quantum chemistry (expressed as an additional text predicate). When performing time-travel search over the Wikipedia history, we find that she was leader of the parliamentary opposition in 2004.

Web tables and microformat data are embedded in HTML pages. Thus, combining structured query predicates with keywords and phrases to be matched in the surrounding text allows much more expressive search. For example, when searching for Germany and Greece over a tables corpus or microformat data, we mostly obtain results on football. If we are actually interested in political relationships, we can add text phrases such as "political negotiations" or "financial crisis" to a combined DB&IR query.

CIDR '13 Asilomar, California USA
6th Biennial Conference on Innovative Data Systems Research (CIDR '13)
January 6-9, 2013, Asilomar, California, USA..

2. USE CASES

Combining RDF-style (SPO) triples with text (X) and time (T) supports use cases from different domains. We sketch some of them, with needs for both *interactive querying* and *analytical reporting.*

Politics. Catchphrases such as "small business owners" or "euro zone stability" are widely used in political speeches. An analyst may want to find all occurrences of such phrases in a Web, news, or social-media corpus and aggregate them by the quoted politician, her/his party, co-occurrences with a particular topic or entity, and so on. A first step could be to formulate/generate a SPOX query combining structured predicates over automatically extracted triples with textual predicates: `?p memberOf ?y . ?y type politicalParty . ?p quotedInMedia ?m {"small business owners"}` where the SPO pattern with the additional text predicate requires phrase matching in the proximity of wherever matching SPO triples were extracted.

Celebrities. Tabloid writers and readers are interested in tracking the lives of entertainment stars. We could analyze how Tom Cruise's romantic affairs (true or speculated) coevolved with his starring in blockbuster movies, and find out what people wrote about these at the time when they were reported. This requires collecting SPOT data (after running extensive IE over (social) media postings) by a query like
`Tom_Cruise actedIn ?m ?t1 . Tom_Cruise romanceWith ?p ?t2`
where `?t1` and `?t2` are time intervals extracted from text, and subsequently joining the resulting quads by a temporal band join on the timespans. This result then has to be extended with text snippets in the proximity of where the matching SPOT quads appear, as a basis for further analysis.

Health. When worried about their health, people tend to consult Dr. Google. Often, though, they are not familiar with medical terminology (e.g., "ulcerative colitis"), but instead use colloquial terms (e.g., "chronic bowel irritation"). Suppose a user wants to find medications for some health issue and learn about their side effects. This could be expressed by the query (via GUI or speech-based UI):

```
?drug medicationFor ?disease {"chronic bowel irritation"}.
?drug hasSideEffects ?x
```

The point here is that the user's vocabulary can be directly used as a text predicate combined with an SPO triple pattern. This is in contrast to an RDF-purist's approach that would require expressing the disease in the explicit terminology of the underlying data or knowledge base (e.g., with a triple pattern like `?drug medicationFor Ulcerative_Colitis`).

Sports. Millions of sports afficionados like to see statistics on the effectiveness of particular player combinations. Given a corpus of SPO-annotated soccer match reports, users may want to know how successful a specific set of midfielders (e.g., Xabi Alonso, Sami Khedira, and Mesut Özil) were in assisting their favorite striker (e.g., Cristiano Ronaldo). This task requires and may need to consider the time dimension as well (e.g., comparing different seasons).

Products. Business analytics on entertainment products has similar requirements. For instance, how do products like Samsung S3 and iPhone 5 perform in the market and compare in consumer opinions? Modern analytics needs to consider consumer comments on review sites and other social-media postings. Again, this requires querying, aggregating, and analyzing SPO triples together with surrounding text and the temporal dimension.

3. ARCHITECTURAL ALTERNATIVES

To manage combined RDF+text+time data and to support the outlined use cases and other applications, various system architectures are conceivable: high-performance systems for SQL, XML, or RDF; NoSQL key-value stores such as HBase and Cassandra; NoSQL document stores, such as MongoDB or CouchDB; queryable map-reduce platforms, such as Hive [12], HadoopDB/Hadapt [2], Hadoop++ [7], or Shark [8]; parallel engines for JSON objects such as ElasticSearch; text engines extended for semistructured data such as Solr; and, of course, hybrid combinations of all these. Each of these design choices comes with its own idiosyncrasies and limitations regarding the combined *functionality & scalability challenge.*

Traditional systems like relational engines with ADTs for textual and spatio-temporal domains or XQuery Full-Text provide sufficiently rich functionality. However, for the kind of schema-free data that we consider (with hundred thousands of different predicates/relations/attributes), they fall short of scaling out to massive data on many machines. RDF engines [1], key-value stores, and Hadoop extensions may scale well for simple workloads like star joins on SPO triples, key-value lookups, key-based grouping, and keyword search, but fail to support complex workloads with long join chains, band joins (e.g., for temporal analytics), or advanced text predicates with composite phrases (e.g., entity names, quotations, slogans), linguistic annotations (e.g., part-of-speech tags), or text proximity search. Extended text engines lack support for joins and other relational operators. Finally, hybrid solutions pose big burdens on application programmers if code needs to be written against federated interfaces.

4. D-HIVE DATA MODEL

In principle, we could adopt a relational data model or an XML-based model, the latter having the advantage of coming with text-document support. However, these are very generic models, not specifically targeted to the kind of data that we consider: SPO triples, keywords and phrases, timestamps. Mapping RDF triples onto relational systems has not been successful, introducing either a super-sparse universal-relation table, or a generic SPO table entailing self-joins for virtually every query, or a physical-design nightmare with clustered tables. Moreover, in our setting, we need to move from triples to quads, quints, etc., with further increase of complexity. The data models that underlie key-value stores or JSON are much simpler, but are very limited in terms of the operations that they support: usually ignoring joins, not to speak of phrase matching, time-travel predicates, etc.

To preserve the schema-free flavor of RDF data, including the use of predicates (relations/attributes) as variables, we decided to start from SPO triples and extend them into quads, quints, etc. This gives a customized multi-dimensional (cube-like) data model that programmers can easily understand. The extra dimensions beyond SPO represent:

- 4: Text keywords associated with an SPO triple, e.g., derived from the Web pages where the triple was extracted from, or referring to the S entity of the triple as part of an entity-specific statistical language model [17].
- 5: To support flexible phrase search (multiple consecutive words or multiple words in proximity) rather than relying on a precomputed fixed set of phrases (which would then be treated as special words), we need to know the position in the associated text where a keyword occurs.

Position indexes are standard in IR engines, but typically come with a major increase in storage and computational costs, compared to vanilla inverted lists for single keywords only.

- 6: The timepoint denoting the begin of the validity of the SPO triple and its associated text (e.g., the begin of a person's term as a CEO).
- 7: The timepoint denoting the end of the validity of an SPO triple and its text.
- 8: The timestamp when the SPO triple was captured ("transaction time").

Note that support for ad-hoc phrases with word positions is crucial to tap into entity names (people, places, products) which are not always in canonical representation (i.e., registered entities in the S or O roles of an SPO triple), marketing slogans, people's quotations, remarks in customer reviews, etc. This aspect has been disregarded by prior work on text cubes (e.g., [6]) and has been way underrated in much of the DB&IR literature. Industrial-strength text engines like those of Google, Bing, etc. or SAP TRex for enterprise search [13] invest great care into phrase indexing.

As for the building blocks of a query language, we start with Sparql-like triple patterns and extend them into multi-dimensional patterns. An oct pattern has the form
`S P O {teXt} @[begin,end] #t`
where each of `S,P,O,teXt,begin,end,t` can be variables (starting with `?`, to be bound by answers) or input values. `teXt` can be a composite predicate `{(word1,pos1),(word2,pos2),...}` with standard phrase search like `{(w1,?p),(w2,?p +1)}` abbreviated as `{"w1 w2"}`. An example is `?s type politician {"quantum chemistry"} @[1980,2000]` (with arbitrary capturing time, hence omitted). Multiple oct patterns or partial octs such as triple, quad, or quint patterns can be conjunctively combined to express joins. For example, we could combine the above search pattern with a second one like `?s bornIn Germany`.

5. D-HIVE ARCHITECTURE

Figure 1 depicts a system architecture that we envision for D-Hive. Queries, posed against the API sketched in the previous section, are first processed by an optimizer, which decides whether a Map-Reduce (MR) execution strategy or a DBMS-style query-processing (QP) strategy is preferable. For instance, for highly selective queries, an MR strategy would be unnecessarily consuming resources, reading from disk and transferring over the network items not needed for the result. With a QP strategy, the execution plan would consult specialized D-Hive indices which can surgically access only the items of interest to the query.

D-Hive indices are either basic indices, such as B-/R-trees and inverted lists per keyword, or more sophisticated indices to handle time intervals, phrases and word proximity, and multi-dimensional queries involving, for example, quints, octs, and more exotic operations like band joins and adjacency joins on time intervals or keyword positions. Indices can also be built to support more efficient processing of MR strategies, along the lines of [2, 7]. The optimizer also decides about which indices to employ and how to process them (e.g., loading small indices in memory, etc).

At the storage layer, D-Hive is designed like a federated system, thus being able to utilize different stores. Note that this federation level is transparent to application programmers; it refers solely to the storage layer and is encapsulated under the D-Hive API. We are currently experimenting with (i) Hadoop DFS, (ii) key-value stores (HBase and Cassandra), document stores (MongoDB), RDBMSs (PostgreSQL) – and plan to consider further variants. The idea here is to replicate the data a modest number of times with different representations for the replicas (using different underlying stores), this way giving more degrees of freedom to the query/analytics optimizer.

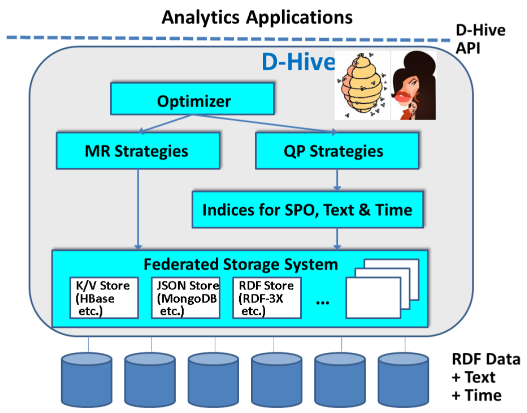

Figure 1: D-Hive System Architecture

6. CHALLENGES

Query Language. We have alluded to extensions of Sparql for querying the combination of RDF, text, and time data, and we have used ad-hoc extensions in examples. However, the design of an appropriate language is a widely open issue and presents the first key challenge. Extending SPO triple patterns with a fourth and fifth dimension of text and time predicates is only one option. The choice of language primitives and composition principles is particularly demanding and critical because queries should be building blocks of more advanced data analytics.

Data Stores. A baseline approach would be to use specialized stores for different dimensions of the D-Hive data items. For instance, an RDF engine (e.g., RDF-3X) may be employed to store SPO triples, text engines (e.g., Lucene) may be utilized for the text parts, and an RDBMS may be used to store time points and intervals (each on all nodes of a cluster). Such an approach would be appropriate if queries involved exclusively either SPO triples, or text phrases, or time predicates. However, for the D-Hive functionality such a fully federated approach is problematic. Going from triples to quads (e.g. with queries involving SPO triples and keywords) would necessitate a large number of needless data accesses and transfers; for instance, the RDF engine's index would not rule out SPO triples not satisfying the keyword constraints and vice versa. Going from quads to quints (e.g. adding capture-time predicates to the query) or to octs (adding time begin and end points for validity and capture times) would greatly exacerbate this problem.

Indexing. Indices may be stored at a different system than the raw data. For instance, a key-value store may be appropriate for indices, in the form of key-value tables.

Index size is a great concern. The simple conceptual jump from triples to quints, and from there to octs and higher-dimensional items, may result in exponentially growing index sizes. Simple strategies employed, for example, by indexing all combinations of S, P, and O (as in RDF-3X [11]), although appropriate for their environment, are clearly inappropriate for D-Hive data. With a realistic number of unique SPO triples upwards of 100 million, a few hundred of keywords and phrases, on average (making up the text part of each SPO fact), hundreds of documents where each SPO triple can occur, and with each document having hundreds of temporal versions (e.g., for longitudinal analytics [15]), D-Hive will have to index hundreds of trillions of data items, requiring hundreds of terabytes or even petabytes. What is clearly needed is a judicious selection of appropriate indices for some of the key combinations of each data items' dimensions and use of appropriate multi-dimensional indices, along with specialized indexes for text and time.

Data Dynamics. D-Hive applications are expected to have high dynamics, mostly in the form of insertions of new data (versions). Accomodating these updates, at potentially high rates, poses demanding constraints on indexing. It is widely open how to reconcile the performance needs of complex queries with the required throughput of new insertions. Note that until just a few years back (i.e., before Google introduced its Percolator architecture), this kind of problem was unsolved already for the much simpler case of text only. The combination of RDF data, text, and time turns the update problem into a major challenge.

Advanced Query Processing. Extended triple patterns like quint or oct patterns are not sufficient to facilitate the envisioned kind of entity-relation-text-time analytics. Obviously, we need to add aggregation and grouping operators. While this is straightforward for the SPO part, the challenge lies in coping with the text and time dimensions as well and designing a seamless suite of operators. In addition, various kinds of special operators are needed to cope with word positions and proximity. Most notably, we consider *band joins* [5], and more specifically the special case of *adjacency joins* (with a band size 1), as an important building block. For example, a phrase search for "quantum chemistry" combined with an SPO pattern can be seen as an adjacency join between two patterns on the word-position dimension. This idea generalizes to proximity search, for example, when we want to accept text like "quantum physics and its role in chemistry" as an approximate match to the above query predicate.

Likewise, n-gram analysis, e.g., for frequent user comments in product reviews or for analyzing text-block copying in social media, needs n-1 band joins on the position dimension. Note that this is a conceptual view, from the perspective of expressing queries and analytic tasks; the implementation may use different operators with specific algorithms. Reconciling the ease of use at the conceptual data model level with an efficient and scalable system level is a critical part of the combined functionality & scalability challenge that we are posing.

Temporal adjacency is another case to consider in more depth, for example, to determine time intervals whose end and begin meet exactly. A query that needs such predicates could be for politicians who start being a company's CEO right after finishing their term in a political office.

Optimization of Execution Plans. Estimating the selectivity of combined SPO+text, SPO+time, or SPO+text +time queries is an entirely new game. For RDF data alone, decent methods for selectivity estimation have been developed, and there are good optimizers for join ordering based on such statistics [11]. However, when going from triple patterns to quint patterns and beyond, there is another leap in complexity that join-order optimizers need to cope with. Keyword selectivities cannot be cast into histograms or similar kinds of statistical synopses, and the support for phrases and word proximity makes this issue even harder. Thus, optimizing and efficiently processing select-join queries over all D-Hive dimensions is already a big challenge. For advanced analytics, with additional operators such as n-gram analysis, we need to understand the interplay in the entire pipeline of data-centric and text-centric operators. Prior work [10, 4] has looked into such issues for the case of rule-based information extraction, but mostly focused on the sweet spot of pushing down selective operators. Our setting is even more challenging.

7. CONCLUSION

We have an early version of D-Hive running and have started to conduct performance studies using the Yago knowledge base (with more than a billion RDF triples) in combination with the history (monthly versions) of the Wikipedia full text. We are convinced that now is the right time to reconsider and advance the cross-pollination of DB and IR methods at Web scale for analytics applications that need to integrate RDF, text, and time.

8. REFERENCES

[1] J. Huang et al.: Scalable SPARQL Querying of Large RDF Graphs. PVLDB 4(11), 2011

[2] A. Abouzeid et al.: HadoopDB: an architectural hybrid of MapReduce and DBMS technologies for analytical workloads. PVLDB 2(1), 2009

[3] Knowledge Extraction Workshop, NAACL-HLT 2012, `http://akbcwekex2012.wordpress.com/program/`

[4] L. Chiticariu et al.: SystemT: An Algebraic Approach to Declarative Information Extraction. ACL 2010

[5] D.J. DeWitt et al.: An Evaluation of Non-Equijoin Algorithms. VLDB 1991

[6] B. Ding et al.: Efficient Keyword-Based Search for Top-K Cells in Text Cube. IEEE TKDE 23(12), 2011

[7] J. Dittrich et al.: Hadoop++: Making a yellow elephant run like a cheetah, PVLDB 3(2), 2010

[8] C. Engle et al.: Shark: fast data analysis using coarse-grained distributed memory. SIGMOD 2012

[9] T. Heath, C. Bizer: Linked Data: Evolving the Web into a Global Data Space Morgan & Claypool, 2011

[10] P.G. Ipeirotis et al.: Towards a query optimizer for text-centric tasks. ACM TODS 32(4), 2007

[11] T. Neumann, G. Weikum: The RDF-3X engine for scalable management of RDF data. VLDB J., 2010

[12] A. Thusoo et al.: Hive - a petabyte scale data warehouse using Hadoop. ICDE 2010

[13] F. Transier et al.: Engineering basic algorithms of an in-memory text search engine. ACM TOIS 29(1), 2010

[14] P. Venetis et al.: Recovering Semantics of Tables on the Web. PVLDB 4(9), 2011

[15] G. Weikum et al.: Longitudinal Analytics on Web Archive Data: It's About Time! CIDR 2011

[16] H. Williams et al.: Fast phrase querying with combined indexes. ACM TOIS 22(4), 2004

[17] C. Zhai: Statistical Language Models for Information Retrieval, Foundations and Trends in IR 2(3), 2008