Position Paper: Scaling-out the NCBO Resource Index Processing and Maintenance

Tomasz Wiktor Wlodarczyk, Paea LePendu, Nigam Shah, Chunming Rong

[1] University of Stavanger, 4036, Stavanger, Norway
[2] Stanford University, USA
{tomasz.w.wlodarczyk, chunming.rong}@uis.no
{plependu, nigam}@stanford.edu

Abstract. In this paper we present main challenges related to scaling-out the NCBO Resource Index further. We look into several recent developments that can relate to those challenges. Finally, we propose a solution that we plan to implement together with a description of an intended evaluation.

Keywords: ontology-based annotation, semantic expansion, data-intensive processing, scalability, search, MapReduce, Hadoop.

1 Introduction

The NCBO Resource Index is a system for ontology based annotation and indexing of biomedical data. The key functionality of this system is to enable users to locate biomedical data resources related to particular concepts. That functionality is based on semantically-enhanced search[1]. The Resource Index currently includes 22 different data resources comprising over 3.5 million data elements resulting in 16.4 billion annotations stored in a 1.5 terabyte MySQL database. The semantic expansion consumes considerable amount of resources in terms of storage and processing power. Based on research in density and evolution metrics it became possible to significantly reduce processing requirements so the Resource Index can be computed in an acceptable time on a single machine [1]. However, when scaling from 22 resources to 100 (or more) some form of distributed processing becomes a necessity. Currently, the amount of indexed resources reaches storage and processing limits of a single machine. Of course, more a powerful machine could be used; however, that does not seem to be a sustainable approach in a longer term.

In recent years data-intensive, non-relational processing has seen significant development starting with Google's MapReduce paper [2] and continuing with

[1] http://www.bioontology.org/resources-index

development of Hadoop[2] and related technologies. It seems natural to investigate those technologies in the context of our problem.

In Section 2 of this paper we name particular challenges that we face while scaling the NCBO Resource Index processing and maintenance. We analyze related work to search for possible approaches to those challenges in Section 3. Finally, in Section 4, we propose architecture of a solution that we plan to implement and evaluate. We conclude the main points in Section 5.

2 Challenges

Storing source data. The data that are indexed can be divided in two major types. The first type is raw data harvested directly from each resource before any processing is done. The second type is data after concept recognition that includes term expansion based on semantics. Storing and maintaining the semantically expanded data is what causes the system to exceed 1.5 terabytes at the moment. As the number and kinds of resources grows, we expect to see a more than 10-fold increase in the storage space requirement, far exceeding the capabilities of typical systems. For example, to include the entire PubMed resource (20M articles) in the Resource Index, it was projected to consume nearly 20TB of disk space. Protection against data lost due to mechanical or program failure will add additional complexity–keeping up to three replicas of the data. Fig 1 (A) illustrates the expected increase in data storage requirements. These estimations do not include possible compression.

Processing source data into the Resource Index. There are three main processing steps required to perform the semantic expansion. The first step is concept recognition, which can be classified as embarrassingly parallel—it can easily be divided among processing nodes. The second step is term expansion based on semantics, where the most computationally expensive process relies mostly on join operations. During that stage, information on ontological distance between concepts is also utilized. For every recognized term, approximately 14 additional terms are also associated. Finally, an inverted index is applied on the entire set of associated terms for efficient search. Unlike concept recognition, the expansion step, i.e. join operations, is not easy to parallelize; however, based on the earlier metrics [1], we believe it is possible to achieve. The main method in earlier metrics was to shard the data by resource, with around 10 tables per each resource, which may need to be re-evaluated.

In general, processing time is expected to increase linearly with the amount of data. The linear increase is in itself not a problem. However, the expected increase in data amount is significant. Processing capabilities of a single machine are already limiting frequency of Resource Index update process and that will of course deteriorate even further with more resources. Therefore, it becomes clear that some form of data-intensive distributed processing is necessary.

[2] http://hadoop.apache.org/

Maintaining the Resource Index. The Resource Index that is used for search purposes is a product of several complex processes. However, that does not provide a full picture of complexity of updating Resource Index. At the same time, Resource Index has to be available for search and it would be beneficial to introduce the update as soon as possible. We need special architecture that will allow for concurrency of updates and searches. That will allow for continuous updates to Resource Index without taking the search offline. Moreover, such architecture should support many concurrent requests without noticeable decrease in performance.

3 Related Work

In this section we present related work that has the potential to help with the aforementioned challenges. The biggest focus is not on scientific publications, but on several open-source projects that could be directly or indirectly applied to our scenario. However, some scientific publications are also mentioned where relevant.

We focus on solutions in so called NOSQL or non-relational domain. An alternative could be a distributed relational database. It would require fewer changes in the current architecture. However, less structured nature of the non-relational solution fits our data better. It also seems that non-relational approach should offer greater flexibility in future.

Our approach in this work focuses on materialization of semantic expansion. That is, data is expanded on disk and queries are directly performed on the data. Query should be understood here as a search query, not as a SQL nor as a SPARQL query. The alternative approach would be to expand the query and perform it on non-expanded data. It would save disk space, but might negatively impact the performance at run time as some query expansions might have significant size. The query-expansion approach is the topic of separate research work.

Storing source data. One of the prominent developments in storage is the Hadoop Distributed File System (HDFS)[3] designed based on Google File System [3]. It is a file system that abstracts data distribution across cluster nodes. At the same time it provides transparent and automatic data redundancy. It provides strong scalability and also allows storing unstructured data. Another development is HBase[4], which is based on the HDFS designed based on Google BigTable [4]. HBase is in fact an ordered multidimensional map. Moreover, the final level of the map is always a timestamp that can be later utilized while reading the data to represent state of data at a particular time in past. Both HDFS and HBase directly support Hadoop MapReduce[5], which we describe in the next subsection.

HyperTable[6] is scalable database also based on BigTable. It can run on top of HDFS and several other file systems. It offers relatively more limited integration with

[3] http://hadoop.apache.org/hdfs/

[4] http://hbase.apache.org/

[5] http://hadoop.apache.org/mapreduce/

[6] http://www.hypertable.org/

other Hadoop subprojects than HBase. Cassandra[7] is a more recent development also inspired by BigTable; however, to a lesser extent. It provides limited support in terms of further data processing.

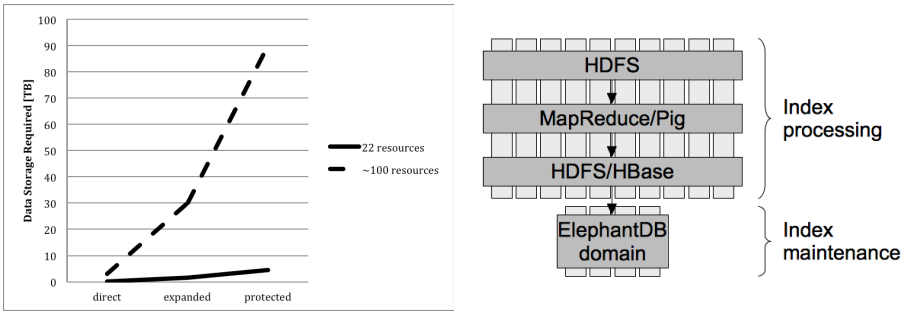

Fig. 1. Current (22 resources) and Expected (100 resources) Data Storage Requirements for the NCBO Resource Index (A). Scheme of Architecture for Scaling-out the NCBO Resource Index Processing and Maintenance (B).

Processing source data into the Resource Index. Processing mechanisms are generally connected to a particular storage system. HDFS and HBase offer probably the biggest variety of choices. First and foremost, they support Hadoop MapReduce which is a generic processing framework based on Google MapReduce [1]. Data processing is performed through java functions. Several different query languages have appeared that base on top of MapReduce. To name a few: Hive[8] is a SQL-like query language, Pig[9] is script-like data processing language and Cascalog[10] is datalog-inspired query language. While Hive and Pig are custom languages Cascalog queries are first-class in Clojure (Lisp dialect that works in Java Virtual Machine).

HyperTable has its own HyperTable Query Language with a SQL-like syntax. Casandra does not offer support of any query language at the moment.

Maintaining the Resource Index. Maintaining the Resource Index includes two main tasks: updating and making it available for rapid search. In the best scenario both tasks should work in parallel. Cassandra and Voldemort[11] can both be considered here as they focus on fast data serving in contrast with bulk processing in, for example, HBase. They can also handle parallel read and writes well. However, they do not offer any dedicated mechanism for handling the connection with the bulk processing backend. Though, it can be constructed. The most recent project that emerged is ElephantDB[12]. It consists of two integrated components where one is dedicated to creating the Resource Index and the other to serving it.

---

[7] http://wiki.apache.org/cassandra/

[8] http://hive.apache.org/

[9] http://pig.apache.org/

[10] https://github.com/nathanmarz/cascalog

[11] http://sna-projects.com/voldemort/

[12] https://github.com/nathanmarz/elephantdb

4 Proposed Solution and Evaluation

It this section we outline a solution we intend to implement and a related evaluation plan. In Fig. 1 (B) we present a sketch of the architecture to be implemented. The main processing will be evaluated on a 10 node cluster, and the query processing part will be implemented on a smaller cluster.

Based on the available solutions we plan to implement data storage both directly in HDFS and in HBase. We will evaluate relative performance differences. In particular, potential benefits of standard and custom HBase index (not to be confused with the Resource Index) and its multidimensional structure. The multidimensional structure of HBase will be also investigated as a way to reduce storage requirements for semantic expansion. In later phases, we will investigate possible applications of time stamping in HBase for analyzing the Resource Index evolution with time. The subject of evaluation will be mostly performance (processing and storage); however, additional elements like ease of maintenance or support for processing approaches will be also assessed. Independently of other tests, compression will be applied to data to examine its impact on both storage requirements and processing time.

The processing stage will be initially implemented in Pig. In later phases, we will compare it with pure MapReduce jobs to potentially leverage a custom HBase index. Finally, we plan to implement the Resource Index maintenance using ElephantDB. To our knowledge this would be first independent evaluation of that system.

5 Conclusions

In this paper we presented challenges related with scaling-out the NCBO Resource Index. A parallel could be found between our challenges and recent developments in data-intensive processing. By looking into those developments we proposed a solution that we plan to implement and evaluate. This should allow scaling-out the NCBO Resource Index to cover all the required resources. In addition, the evaluation will fill gaps in knowledge about relative (processing and storage) performance of the aforementioned technologies.

References

1. P. LePendu, N. F. Noy, C. Jonquet, P. Alexander, N. H. Shah, M. A. Musen, Optimize First, Buy Later: Analyzing Metrics to Ramp-up Very Large Knowledge Bases, 9th International Conference on Semantic Web (ISWC 2010), Shanghai, China, Springer, 2010.
2. J. Dean and S. Ghemawat, MapReduce: Simplified Data Processing on Large Clusters, OSDI'04, San Francisco, CA, December, 2004.
3. S. Ghemawat, H. Gobioff, and S. Leung, The Google File System, 19th ACM Symposium on Operating Systems Principles, Lake George, NY, October, 2003.
4. F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, Bigtable: A Distributed Storage System for Structured Data, OSDI'06, Seattle, WA, November, 2006.