Surfing Big Data Warehouses for Effective Information Gathering

Nunziato Cassavia[1], Pietro Dicosta[4], Elio Masciari[1] and Domenico Saccà[2,3]
[1]*ICAR-CNR, Rende, Italy*
[2]*DIMES UNICAL, Rende, Italy*
[3]*Centro di Competenza ICT-SUD, Rende, Italy*
[4]*NTT DATA, Rende, Italy*

Keywords: Big Data, Solr, Clustering.

Abstract: Due to the emerging Big Data paradigm traditional data management techniques result inadequate in many real life scenarios. In particular, OLAP techniques require substantial changes in order to offer useful analysis due to huge amount of data to be analyzed and their velocity and variety. In this paper, we describe an approach for *dynamic* Big Data searching that based on data collected by a suitable storage system, enriches data in order to guide users through data exploration in an efficient and effective way.

1 INTRODUCTION

Nowadays, the availability of huge amounts of data from heterogeneous sources, exhibiting different schemes and formats and being generated at very high rates, led to the definition of new paradigms for their management – this problem is known with the name *Big Data* (uno, 2008; due, 2010; tre, 2011; Agrawal et al., 2012; Lohr, 2012; Manyika et al., 2011; Noguchi, 2011a; Noguchi, 2011b; Labrinidis and Jagadish, 2012). As a consequence of new perspective on data, many traditional approaches to data analysis result inadequate both for their limited effectiveness and for the inefficiency in the management of the huge amount of available information. Therefore, it is necessary to rethink both the storage and access patterns to big data as well the design of new tools for data presentation and analysis. In particular, OLAP analysis tools require suitable adjustments in order to work for big data processing effectively. Indeed, it is crucial, during the construction and analysis of a data warehouse, to exploit ad-hoc tools that allow an easy and fast search of data stored in several nodes distributed over the storage layer.

More in detail, while building a data warehouse for Big Data, the key to a successful analysis (i.e. a fast and effective one) is the availability of good indexing mechanisms. Therefore, an additional cost in terms of storage space consumption needed for storing the appropriate indices is to be taken into account.

It is worth noticing that the problem of fast accessing relevant pieces of information arises in several scenarios such as world wide web search, e-commerce systems, mobile systems and social networks analysis to cite a few.

Successful analyses for all the application contexts rely on the availability of effective and efficient tools for browsing data so that users may eventually extract new knowledge which s/he was not interested initially.

In this paper, we shall describe the architecture of a system for full-text search, capable to operate over Big Data and offering the chance to "surf" the data in a simplified manner, while keeping traditional operators available in an OLAP based system such as roll-up, drill-down, slice and dice. Moreover, we overcome limitations of traditional OLAP analysis systems, as in our system analysis dimensions are not limited to those defined a priori by the warehouse architect, but they are dynamically added by inducing them from the data. This enrichment of the initial dataset is referred to as Data Posting and can be achieved by suitable data mining techniques, either in batch mode (i.e. taking into account the whole dataset) or on-the-fly by limiting the analysis to the result of current search executed by users.

Our System in a Nutshell. Building a system for Big Data management is a complex activity as many architectural choices are affected by the data at hand. In this respect, our system is quite intriguing as we exploited several utilities in order to make the data analysis step easier also for non expert users. Moreover, we built our prototype using quite powerful tools that are either open source or freely available for research purposes.

First of all, we used *Hbase* as the basic tool for data management as it is highly scalable and fault-tolerant – Hbase is an open source, non-relational, distributed database system, modeled after Google's BigTable and developed as part of Apache Software Foundation's Apache Hadoop project. Moreover, as it is flexible as not tied to a fixed scheme, we can add new columns (attributes) to predefined family column tables without any interruption in service delivery. The latter feature has been crucial for data enrichment as it could happen that mining results could affect only a limited portion of the overall database. Moreover, it enables fast reading of data, thus making the querying phase quite efficient. Unfortunately, for full-text search, response times are still too high thus limiting the usability of the system; therefore, we implemented a specialized index by using *Solr*, which is an open source enterprise search platform from the Apache Lucene project, whose major features include full-text search, hit highlighting, faceted search, real-time indexing, dynamic clustering, database integration, NoSQL features and rich document (e.g., Word, PDF) handling.

The Solr's feature of supporting faceted navigation turned out to be very useful for our purposes: facets are generated for modeling data dimensions that allow users to drill-down or roll-up data. Furthermore, facets allow user suggestions based on previous search performed. We fully exploited the *SolrCloud* release that enjoys enhanced highly scalable and fault tolerant features and empowers distributed indexes as it is based on *HDFS* as file system. We used *Zookeeper* (another software project of the Apache Software Foundation), for enabling distributed configuration and synchronization services as well as naming registry for large distributed systems. We pinpoint that, as Hbase uses the same services, the overall architecture turned out to be rather powerful and flexible.

As we want to achieve full integration between Hbase and SolrCloud, we need to take into account multivalued attribute management. Indeed, SolrCloud provides a support for multivalued storage, whereas for Hbase we need to suitably pre-elaborate them (e.g., by adding a colum suffix). As an example, consider a hotel having several email contacts. Using Hbase we can model this as follows:

$columnFamily\_anagrafic\_info[email_1 :< value1 >, email_2 :< value2 >, \cdots$

$email_n :< valuen >]$.

On the contrary, SolrCloud allows the definition of a multivalued field as follows:

$< fieldname = \text{“}email\text{”}type = \text{“}string\text{”}indexed = \text{“}true\text{”}multivalued = \text{“}true\text{”}stored = \text{“}true/ >$

In order to guarantee full integration of both systems we need to provide a mapping between the two systems that can be performed by *Morphline*, a new command-based framework that simplifies data preparation for Apache Hadoop workloads. The configuration file (named morphline.conf) will contain commands like the one reported in the following:

$extractHBaseCells\{mappings : \{inputColumn : \text{”}columnFamily\_anagrafic\_info : email\text{”}outputField : \text{”}email\text{”}type : stringsource : value\}$

In order to speed-up the index construction we exploit map-reduce as it allows the batch construction of the overall index by accessing all nodes in the cluster.

However, in some application scenarios (e.g. monitoring systems) we need a (near) real time indexing that can be done by the *Lily* Indexer that provides the ability to quickly and easily search for any content stored in HBase by indexing HBase rows into Solr, without writing a line of code. Indeed, it is fully complaint with several extraction tools like *Flume* that is a distributed Apache servicee for efficiently collecting, aggregating and moving large amounts of streaming data flows so that data are available for searching immediately after their insertion in the data storage layer.

2 BACKGROUND ON COMPLEX SEARCHING

A typical example of system devoted to complex data querying is represented by search engines. The results of a search returned by the engine cannot be considered as a custom map built by query results but, based on them, additional knowledge about data being queried can be learnt by iterative refinement of search dimensions and parameters as reported in Figure 1.

In this process, the type of research being performed has to be taken into account. Indeed, there is a big difference between the simple search of well-defined terms and the dynamic learning by exploratory research. Obviously enough, in the first case, a search engine such as Google, is able to give

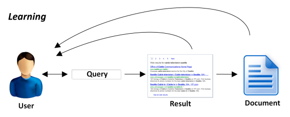

Figure 1: Learining By Results.

results in less than a second. As a matter of fact, due to its quick result presentation, many users go through Google even if they know the URLs of the resources that are interested in.

However, in some cases users do not know exactly how to find the desired information about an object (e.g. a book). In this case the Amazon model depicted in Figure 2 is more suitable. More in detail, Amazon search tools, includes the product categories and some recommender systems, making the user search experience quite interactive and iterative. In a sense, intermediate results guide users to a better definition of target information.

Furthermore, search engines usually allow non-structured queries (known as "ranked retrieval") whose results are sorted by some relevance rank disregarding their precision. As a matter of fact, this kind of queries is easier to pose by users compared to boolean expressions, but they can cause the setting of a not precise evaluation criteria for document comparison.

As a possible improvement, some search engines like Yahoo! Directory, offer context search. More in detail, the contents of a directory is organized hierarchically, the user is guided to a subset of documents potentially related to information being searched avoiding the possibility to pose free text queries. In this respect, users re-think and refines their needs, thus learning the adjustments to the search being performed by exploiting the available choices. To better understand how directory navigation works, consider how accommodation booking portals work. They offer a hierarchical navigation systems, i.e. from the home page, user can choose the desired country, then s/he can specify the city and finally the type of structure s/he is interested in. This navigation model suffers a great limitation due to taxonomy specification. Indeed, taxonomy specified by the service designer may not meet user needs.

2.1 The Real-time *faceted* Navigation

It is possible to overcome the limits of the types of search tools explained above by *faceted* navigation that helps users in the information "surfing" process. Consider the faceted view of a search engine depicted in Figure 3. Starting the search from the home page, the user has the opportunity to search information about the country *and* about several attributes of the structures pertaining to the search. For example, s/he can browse city ($Cosenza, Scilla, \ldots$)), the structure type ($Hotel, B\&B, apartment, \ldots$), their rating $(2, 3, 4, \ldots)$. As a feature is selected, the user can refine the search by selecting another attribute among those available for the current selection. During the browsing process, it is also possible to discard features no longer relevant to the search (i.e. s/he can perform *dimensional filtering*). This iterative process guides the user through the accommodation search by selecting a custom path instead of a hierarchy provided by the service designer.

We stress that efficient faceted navigation (i.e. easy to use and providing access to richer information) relies on the availability of some features that are common to the objects being searched. In a sense, it is impossible to implement faceted navigation for a site that sells products not sharing at least a category.

The faceted search pattern described above can be enhanced by exploiting data mining approaches for information enrichment. To this end we propose a novel approach to Data Posting, i.e. based on raw data and ontologies we can add new dimensions induced by analyzing search queries and results. More in detail, we store query result as a materialized data cube to be used for further search. These data will be used as training set for an clustering algorithm that will group query results in a unsupervised way. The obtained clustering will be used for extracting features relevant to the query that have not been specified by the user neither have been considered for building the query result. As an example consider a user searching for a restaurant in Rome. S/he will type the query "restaurant in Rome" (also many search engines will suggest this statement). Traditional search results then will include restaurants located in the city along with their rank. Indeed, by exploiting our approach we are able to suggest users a further interesting parameter (i.e. analysis dimension) as the rank of apetizers, main courses and sweets thus allowing a more focused search.

Data Posting was first introduced by (Saccà

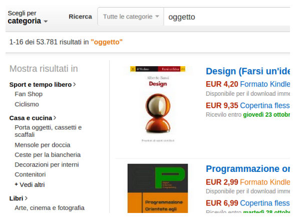

Figure 2: Amazon search.

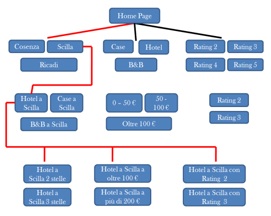

Figure 3: Faceted navigation example.

and Serra, 2013). The *data posting setting* $(\mathbf{S}, \mathcal{D}, T, \Sigma_{st}, \Sigma_t)$ consists of a source database schema S, a domain database scheme $\mathcal{D}$, a target flat fact table T, a set Σ_{st} of source-to-target count constraints and a set Σ_t of target constraints. The *data posting problem* associated with this setting is: given finite source instances I_S for $\mathbf{S}$ and $I_{\mathcal{D}}$ for $\mathcal{D}$, find a finite instance I_T for T such that $\langle I_S, I_{\mathcal{D}}, I_T \rangle$ satisfies both Σ_{st} and Σ_t.

The main difference w.r.t. classical data exchange is the presence of the domain database scheme that stores "new" values (dimensions) to be added while exchanging data. The actual values to be assigned to dimensions are defined by means of the target constraints. In a motto we can say that "data posting is enriching data while exchanging them". Next section is devoted to the description of our prototype for Data Posting.

3 A SYSTEM FOR BIG DATA SEARCH

Our system, developed as part of *DICET-INMOTO* project, is tailored for providing users a flexible tool for full-text search, that is interactive, scalable on Tourism Big Data and dynamic. To this end we need to exploit suitable indexing and data management strategies. It is based on several open source tools as Apache Hadoop, Flume, HBase, Solr, Lily HBase Indexer and Hue supporting our Data Posting strategy as depicted in Figure 4.

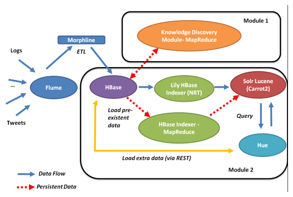

Figure 4: System Architecture.

As tourist big data arrive in a streaming way, we need to properly collect them by Flume that is a reliable and distributed service to efficiently collect, aggregate and forward huge amounts of data. It offers a flexible architecture for data streaming providing a fault tolerant system based on a configurable reliability mechanism. Once data are gathered by Flume module, they are pre-elaborated "on the fly" by Morphline and stored in a data warehouse stored on HBase. Morphline module is devoted to data cleaning and data mapping on the column set in the datastore.

For querying purposes we provide two indexing strategies, *static* and *dynamic*. We provide both features in order to deal with all the possible use cases. More in detail, if data pertaining the query have been stored in the data warehouse static indexing turns to be more effective. We perform this operation by Map Reduce Indexer that takes advantage of the clustered structure of the datastore. On the contrary, if new data that have to be inserted into the data warehouse, we take advantage of near real time indexing provided by Lily framework.

In order to allow efficient on line analysis when performing full-text search, we exploit Apache Solr system. It allows searching for keywords in any field that was previously indexed and allows to display faster the documents matching the query. Moreover, Solr allows several useful operations as field facets, range queries and pivot facets, that can be used for providing user the classical OLAP operators (slice & dice, drill-down, roll-up, pivoting) thus making Solr an excellent real-time analysis engine for text documents.

As an example, in a website, the log files and additional information on user behavior can be indexed by Solr in order to allow (timed) range queries for a keyword. It is also possible to build information graphs containing aggregate information, such as the growth over time of the number of registered users or transactions aggregated by type.

Furthermore, we also exploit Carrot2 (a Solr plugin) in order to make the search even more effective as it provides real time clustering features that are exploited to derive new dimensions for analyzing data. More in detail, based on the clusters obtained by Carrot2, we add new categories to the data warehouse that will be exploited for guiding user through the search.

In order to display search results, we exploit Hue features. The latter is a software that perfectly fits the Hadoop ecosystem. It offers a user friendly interface shown in Figure 5, that can be customized for different user categories, e.g. end-users and domain experts. In particular, end-users are allowed to search only data indexed by Solr, while domain experts may also view/edit data available in the data warehouse (including those induced by the system automatically) and add new dimensions.

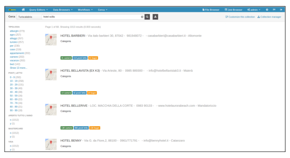

Figure 5: Hue interface.

For the sake of efficiency, we prevent the reversal of the entire data warehouse within Solr. More in detail, we distinguish between indexed data used to search for documents and data stored on Solr which can be accessed directly avoiding the access to raw data. In order to improve system performances we keep the minimum amount of information on Solr index while we allow access to the complete information by REST API service[1].

Finally, the architecture reported in Figure 4, offers two solutions for the different data to be managed: *persistent data* and *streaming data*. In Figure 4 we denote by blue full arrows the components that are used for data streams processing, while the ones denoted by red dashed arrows deal with persistent data.

Indeed, the overall architecture is composed of two modules: *Module 1* for *Near Real Time* processing and *Module 2* for *Batch* processing whose features are described below:

1. *Near Real Time* processing is tailored for end-users. This module allows:
   - Full-text search, interactive, scalable and flexible indexing system;
   - Discovering new dimensions of analysis based on result search logs;
   - Static faceted navigation of categories defined a priori and dynamic faceted navigation by exploiting dimensions induced on line by clustering algorithms.
2. *Batch* processing is tailored for expert users. This module allows:
   - Off-line discovery (i.e. based on the whole dataset) and storage of new dimensions for the data warehouse by a customizable result visualization;
   - Selection of the mining tool for data analysis (we actually implemented clustering and classification features) based on the scenario to be analyzed;

4 CONCLUSION

Big data analysis is a challenging task as we need to take into account the velocity, variety and volume of information to be analyzed. Indeed, such features heavily influence the design of a system for big data analysis. In this respect, we analyzed several design options in order to implement a prototype for Big Data Warehousing offering advanced search functions. Our prototype has been used for tourism big data analysis both by end-users and domain experts. Results on the usability of the system were quite satisfactory. We are now gathering real data form public sources (Facebook, Twitter, Yelp, Tripadvisor) in order to perform a detailed analysis of the accuracy we can obtain by our prototype.

[1]We allow access by HBase Rest Server

ACKNOWLEDGEMENTS

This work was supported by MIUR Project PON04a2_D DICET INMOTO Organization of Cultural Heritage for Smart Tourism and REal Time Accessibility (OR.C.HE.S.T.R.A.)

REFERENCES

(2008). Big data. *Nature*.

(2010). Data, data everywhere. *The Economist*.

(2011). Drowning in numbers - digital data will flood the planet - and help us understand it better. *The Economist*.

Agrawal et al., D. (2012). Challenges and opportunities with big data. A community white paper developed by leading researchers across the United States.

Labrinidis, A. and Jagadish, H. V. (2012). Challenges and opportunities with big data. *PVLDB*, 5(12):2032–2033.

Lohr, S. (2012). The age of big data. *nytimes.com*.

Manyika, J., Chui, M., Brown, B., Bughin, J., Dobbs, R., Roxburgh, C., and Byers, A. H. (2011). Big data: The next frontier for innovation, competition, and productivity. *McKinsey Global Institute*.

Noguchi, Y. (2011a). Following digital breadcrumbs to big data gold. *National Public Radio*.

Noguchi, Y. (2011b). The search for analysts to make sense of big data. *National Public Radio*.

Saccà, D. and Serra, E. (2013). Data posting: a new frontier for data exchange in the big data era. In Bravo, L. and Lenzerini, M., editors, *Proceedings of the 7th Alberto Mendelzon International Workshop on Foundations of Data Management, Puebla/Cholula, Mexico, May 21-23, 2013.*, volume 1087 of *CEUR Workshop Proceedings*. CEUR-WS.org.