2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing

Supporting Queries and Analyses of Large-Scale Social Media Data with Customizable and Scalable Indexing Techniques over NoSQL Databases

Xiaoming Gao
School of Informatics and Computing
Indiana University
Bloomington, IN, USA
gao4@indiana.edu

Judy Qiu
School of Informatics and Computing
Indiana University
Bloomington, IN, USA
xqiu@indiana.edu

***Abstract*— Social media data analysis demonstrates two special characteristics in Big Data processing. First, most analyses focus on data subsets related to specific social events or activities instead of the whole dataset. Second, analysis workflows consist of multiple stages, and algorithms applied in each stage may use different computation and communication patterns depending on processing frameworks. This paper presents our efforts in supporting the data storage and processing requirements for such characteristics. To achieve efficient queries about target data subsets, we propose a general customizable and scalable indexing framework that can be built over distributed NoSQL databases. This framework allows users to define suitable customized index structures for their query patterns against social media data, and supports scalable indexing of both historical and streaming data. We implement this framework on HBase, and name it IndexedHBase. Starting from IndexedHBase, we build a distributed analysis stack based on YARN to support analysis algorithms using different processing frameworks, such as Hadoop MapReduce, Harp, and Giraph. This analysis stack is used to host the Truthy social media data observatory, and we have applied the customized index structures in supporting both query evaluation and sophisticated analysis algorithms. Performance tests show that our solutions outperform implementations using both direct raw data scans and current indexing mechanisms in existing NoSQL databases.**

Keywords-Social media data analysis; customizable and scalable indexing; NoSQL databases; YARN;

I. INTRODUCTION

As data intensive applications evolve, many research projects involving Big Data require efficient extraction and analysis of specific data subsets, rather than the whole dataset. Social media data analysis is one such example. While social media platforms like Twitter provide vast data about people's social life, most research analyses focus on data subsets related to specific social events or activities: congressional elections [6], protest events [4], etc. Compared with the sheer size of the entire dataset at TB or PB level, the subsets are often smaller by orders of magnitude. For such scenarios, efficient query mechanisms are needed for limiting analysis computation to the exact scope of the target subsets.

A closer observation of social media data reveals features including high-volume streaming data, fine-grained data records for social updates, and written-once-and-read-many type of access patterns. These features suggest distributed NoSQL databases as good options for the storage solution. However, the level of indexing support varies significantly across different NoSQL databases, and their current index structures are not flexible enough to handle the queries in social media data analyses [12]. To solve this problem, we propose a general customizable indexing framework that can be built over distributed NoSQL databases. This framework allows users to define customized index structures that contain the exact necessary information about the original data so as to achieve efficient queries about social events. By choosing proper mappings between the abstract index structures and the storage units of the underlying NoSQL database, scalable indexing of historical and streaming data can be achieved. We implement this framework on HBase [3], and release it as an open source project called IndexedHBase [16].

Another important characteristic of social media data analysis is that the analysis workflows normally consist of multiple stages, each with a diversity of algorithms to process the target data subsets, as illustrated in Figure 1. Different algorithms may demonstrate various computation and communication patterns that are suitable for different processing frameworks. Therefore, to achieve efficient overall execution of the workflows, we extend IndexedHBase to build an analysis stack based on YARN [23], which is specially designed for dynamic scheduling of multiple analysis tasks using different parallel processing frameworks in a shared environment. This analysis stack is used to support the Truthy social media data observatory [17], and we have developed multiple parallel algorithms as basic building blocks for constructing analysis workflows. Our experience shows that the customized indices are valuable for developing both efficient query evaluation strategies and various post-query analysis algorithms.

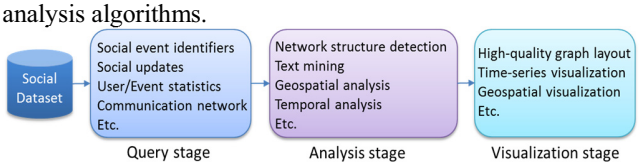

Figure 1. Stages in a social media data analysis process.

The rest of this paper is organized as follows. Section 2 investigates related work. Section 3 explains the abstract data model and index structure of our customizable indexing framework and describes our implementation on HBase. Section 4 tests our architecture with the Truthy application. Section 5 gives the performance evaluation results of some typical queries and analysis tasks. Section 6 concludes and discusses potential future work.

978-1-4799-2784-5/14 $31.00 © 2014 IEEE
DOI 10.1109/CCGrid.2014.57

II. RELATED WORK

Hadoop++ [7], HAIL [8], and Eagle-Eyed Elephant [10] are systems that try to extend Hadoop [2] with indexing mechanisms to facilitate MapReduce queries. However, they schedule MapReduce tasks based on data blocks or splits on HDFS, and tasks may have to scan irrelevant data during query evaluation. In contrast, our framework does indexing on the data record level, and query and analysis tasks need only access relevant data records to produce the result.

HBase [3], DataStax (Cassandra) [5], MongoDB [18] and Riak [21] are examples of distributed NoSQL databases with varied levels of indexing support. Our customizable indexing framework can be implemented on these systems to extend their existing indexing functionality with more flexibility for handling the query patterns from different applications.

HIndex [15] is a project that uses coprocessors to build secondary indices on HBase. It achieves a short latency for index updates by collocating the regions of index tables and data tables. Compared with HIndex, our framework puts an emphasis on the customizability of index structures and general applicability over most NoSQL databases.

III. CUSTOMIZABLE INDEXING FRAMEWORK

A. Input Data Model

The customizable indexing framework uses the concept of **data record** and **record set** to model the input data to be indexed. A **record set** is composed of zero to multiple **data records.** Each **data record** can be modeled by a JSON type of nested key-value pair list data structure uniquely identified by an "**id**" field, as shown in Figure 2. These concepts can be easily mapped to the data storage units of various NoSQL databases. For example, a **record set** can be implemented as a table in HBase, a bucket in Riak, or a collection in MongoDB. Correspondingly, a **data record** can be implemented as a row in HBase, an object in Riak, or a document in MongoDB.

```
{
  "id":34077,
  "text":"Enjoy the great #euro2012",
  "created_at":"2012-06-12 23:22:16",
  "geo":{
          "type":"Point",
          "coordinates":[-6.2219492,
                         52.8767352]
  },
  ...
}
```

Figure 2. An example of nested data model for input data records.

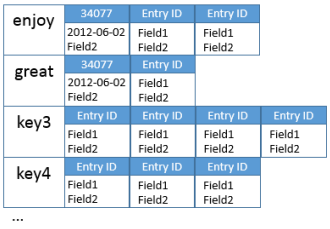

Figure 3. Abstract index structure.

B. Abstract Index Structure

Figure 3 illustrates the abstract index structure used by our framework. The overall structure is organized by a sorted list of **index keys**, and each **index key** can be associated with a varied number of **index entries**. Each **index entry** contains a unique **entry ID**, and a varied number of **entry fields** for embedding additional information about the indexed data. **Index entries** are sorted by **entry IDs**. This structure is similar to the posting lists used in inverted indices [25], but the major difference is that our framework allows users to customize what to use as **index keys**, **entry IDs**, and **entry fields** through an index configuration file, as illustrated in Figure 4. Users can also define their own functions (UDFs) for generating **index keys** and **entries** for a given data record. By using proper index configurations or UDFs, it is possible to create various index structures, such as single dimensional index, multi-dimensional index, inverted index for text retrieval, or even the geospatial index described in [19].

```
<index-config>
  <source-set>tweets</source-set>
  <source-field>text</source-field>
  <source-field-type>full-text</source-field-type>
  <index-name>textIndex</index-name>
  <index-entry-id>{source-record}.id</index-entry-id>
  <index-entry-field>{source-record}.created_at</index-entry-field>
</index-config>
<index-config>
  <source-set>users</source-set>
  <index-name>snameIndex</index-name>
  <indexer-class>iu.pti.hbaseapp.truthy.UserSnameIndexer</indexer-class>
</index-config>
```

Figure 4. An example index configuration file.

C. Interface to Client Applications

Figure 5 presents the major operations provided by our customizable indexing framework to client applications. The client application can use a **general customizable indexer** to index or un-index a data record. The **general customizable indexer** analyzes the index configuration file and generates index entries for the data record, invoking a user-defined indexer if necessary. Then the insertion or deletion of index entries is translated into data operations supported by the underlying NoSQL storage substrate for real execution. To complete a search using an index structure, the client application can invoke a **basic index operator** provided by the framework, or a **user defined index operator**. Multiple constraints can be specified as parameters to filter the index entries by their keys, entry IDs, or entry fields. Constraint types currently supported are **value set constraint**, **range constraint**, and **regular expression constraint**.

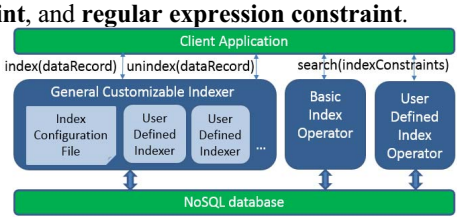

Figure 5. Interface to client applications.

D. Implementation on HBase

Regarding implementation of the customizable indexing framework, our key observation is that most existing NoSQL databases already support scalable and efficient data access through their respective data models. Therefore, by defining a proper mapping between the abstract index structures and the actual data models of the NoSQL databases, it is possible to leverage their existing data distribution and load balancing mechanisms to achieve scalable indexing for our framework. Our current implementation uses HBase as the storage substrate. Specifically, we use a table to implement an index structure, a row key for an index key, a column name for an

entry ID, and a column value for all the entry fields. Since table data is stored under the hierarchical order of <row key, column name, timestamp>, it is easy to support range scans over the index keys or entry IDs. Moreover, based on the region split and load balancing mechanisms provided by HBase, we are able to achieve efficient and scalable real-time indexing of streaming data. For more discussion about the cost of online index building and dynamic changes of the index structures, please refer to [12].

IV. Case Studies with Truthy

Truthy is a social media data observatory designed for analysis and visualization of information diffusion on Twitter. It collects data through the Twitter streaming API [22]. The size of historical data in the format of .json.gz files is more than 10 terabytes. The current data rate of the dynamic stream is 45-50 million tweets per day. Tweets come in the form of structured JSON strings that contain information about users, their tweets, and the "retweet" relationship among tweets.

Truthy uses the concept of "meme" to represent a set of related posts corresponding to a specific discussion topic, communication channel, or social event/activity. Memes can be identified through elements contained in the text of tweets. These include *keywords*, *hashtags* (e.g. #euro2012), *user-mentions* (e.g. @youtube), and *URLs*. Most analysis workflows start with queries about memes [12], e.g. ***get-mention-edges(memes, time-window).*** Here ***memes*** is a list of meme identifiers such as hashtags, and ***time-window*** is given as a pair of time points like [2012-06-08T00:00:00, 2012-06-23T23:59:59]. This query first finds all the tweets containing the given meme identifiers within the ***time-window***, and then extracts all the user-mention edges contained in these tweets. These edges as a whole construct a user-mention network for the given ***memes***, representing a typical pattern of information diffusion on social networks.

We test IndexedHBase with the Truthy queries, designing a set of multi-dimensional index structures. Figure 6 illustrates one of them. This structure indexes both text and non-text data fields, but does not store any frequency or position information about the indexed terms. This is because the Truthy queries are not designed for retrieving the top-N most related documents, but for extracting information such as user-mention networks from all related social updates. To the best of our knowledge, this index structure is not currently supported by any existing NoSQL databases.

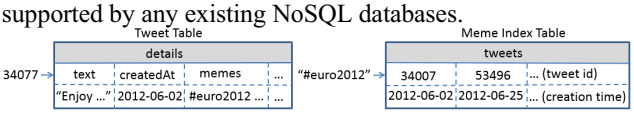

Figure 6. Example data and index tables designed for Truthy.

To achieve efficient execution of analysis workflows in Truthy, we upgraded our previous analysis stack in [13] to a new one based on YARN [23], as shown in Figure 7. The **Indexing Module** provides efficient and scalable customized index building for both streaming and historical data. The **Query Analysis Engine** completes queries about interesting data subsets through index operators, and dynamically invokes different parallel processing frameworks to execute analysis tasks over the query results.

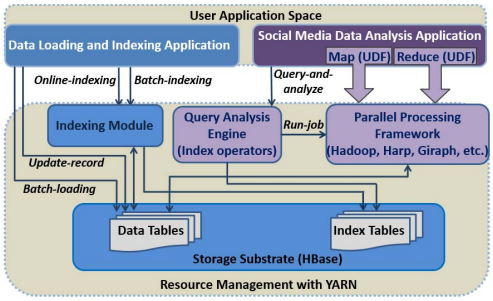

Figure 7. Architecture of data analysis stack based on YARN.

Based on this analysis stack, we developed the following building blocks wrapped up with shell scripts for constructing analysis workflows:

A two-phase parallel query evaluation strategy for all the queries used in Truthy, as described in [12].

A parallel related hashtag mining algorithm implemented with Hadoop MapReduce [2], as described in [13]. The advantage of this algorithm is that it only uses a small portion of the original data, and relies on the indices to complete the major part of computation. Since the size of index data is significantly smaller, this algorithm is much faster than a solution based on original data scanning.

A parallel algorithm for generating the daily frequencies of all hashtags during a given time window. This is useful for many analysis purposes, such as generation of meme evolution timelines [4] and meme lifetime distribution [24]. Considering the schema of "Meme Index Table" in Figure 6, it is obvious that this can be done solely by scanning the index without touching any original data. The algorithm is implemented as a Hadoop MapReduce program. Each mapper takes one region of the "Meme Index Table" as input and generates the daily frequencies for each hashtag by going through the corresponding row and simply counting.

An iterative MapReduce implementation of the Fruchterman-Reingold algorithm [11] for graph layout generation. This algorithm is useful in visualizing many graph structures, such as retweet networks and user-mention networks. We have upgraded the implementation in [13] with Harp [14], the new YARN-compatible version of Twister [9].

In addition, **a parallel version of the label propagation algorithm** [20] using Giraph [1] is under development.

V. Performance Evaluation

This section presents the results of several important performance evaluation tests, which clearly demonstrate that our customizable index structures can greatly improve the efficiency of query evaluation and analysis tasks in Truthy. All tests are done on a private eight-node cluster. The hardware configuration of each node is listed in Table I. Each node runs RHEL 6.5 and Java 1.7.0_45. For the deployment of YARN and IndexedHBase, Hadoop 2.2.0 and HBase 0.96.0 are used. For the deployment of Riak, each node runs Riak 1.2.1, using LevelDB as the storage backend.

Figure 8 shows the performance comparison of three query evaluation strategies for a typical query in Truthy. The **Hadoop-FS** strategy uses a MapReduce program to scan the

.json.gz files (one file for each day), and process the matched tweets to complete query. **Riak** represents a strategy using the text indices with "inline fields" supported by Riak, as described in [12]. The results clearly demonstrate that **IndexedHBase** is significantly faster than the other two strategies. Furthermore, the difference between **Riak** and **IndexedHBase** gets bigger as the time window gets longer, suggesting that **IndexedHBase** is especially good at queries with large intermediate data and result sizes.

Figure 9 compares the performance of our solutions on **IndexedHBase** against two **Hadoop-FS** implementations in completing two analysis tasks. The first task mines related hashtags for "#p2" using data between 2012-09-24 and 2012-11-06. The second task generates daily meme frequencies of all hashtags for 2012-06. The **Hadoop-FS** implementations use MapReduce programs to scan the corresponding files and generate the results. Again, our solutions are significantly faster by factors of ten. More importantly, this comparison clearly demonstrates the value of indices in supporting analysis tasks beyond the basic queries.

TABLE I. HARDWARE CONFIGURATION OF EACH NODE

CPU	RAM	Hard Disk	Network
4 * 4 Quad-Core AMD Opteron 8356 2.3G Hz	16GB	4 TB	1Gb Ethernet

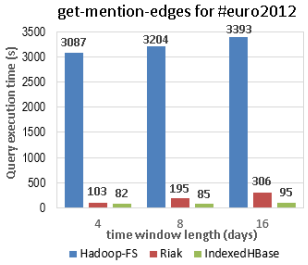

Figure 8. Results for query evaluation tests.

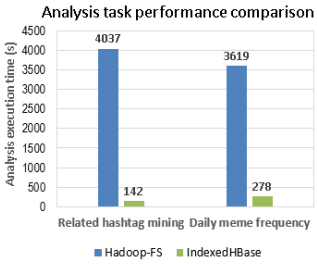

Figure 9. Results for analysis task performance tests.

VI. CONCLUSIONS AND FUTURE WORK

Three major conclusions can be drawn from our experience in this project. First, to achieve the optimal query evaluation performance for different domain applications, index structures should be customizable rather than static. Second, indexing is valuable for not only query evaluation, but also analysis and mining tasks. Lastly, an extendable analysis stack that can dynamically adopt different processing frameworks to handle different tasks is crucial for efficient execution of social media data analysis workflows. We look forward to exploring the value of customized indices to support streaming analysis algorithms in the future.

ACKNOWLEDGEMENTS

This research is in part supported by National Science Foundation CAREER Award OCI-114943 and DARPA (grant W911NF-12-1-0037).

REFERENCES

[1] Apache Giraph. https://giraph.apache.org/.

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

[3] Apache HBase. http://hbase.apache.org/.

[4] Conover, M., Ferrara, E., Menczer, F., Flammini, A. The Digital Evolution of Occupy Wall Street. PLoS ONE, 8(5), e64679. 2013.

[5] DataStax. http://www.datastax.com/.

[6] DiGrazia, J., McKelvey, K., Bollen, J., Rojas, F. More Tweets, More Votes: Social Media as a Quantitative Indicator of Political Behavior. Available at SSRN: http://dx.doi.org/10.2139/ssrn.2235423. 2013.

[7] Dittrich, J., Quiané-Ruiz, J., Jindal, A., Kargin, Y., Setty, V., et al. "Hadoop++: making a yellow elephant run like a cheetah (without it even noticing)," Proc. VLDB Endow. 3, 1-2 (Sep. 2010), 515-529.

[8] Dittrich, J., Quiané-Ruiz, J., Richter, S., Schuh, S., Jindal, A., et al. "Only aggressive elephants are fast elephants," Proc. VLDB Endow. 5, 11 (Jul. 2012), 1591-1602.

[9] Ekanayake, J., Li, H., Zhang, B., Gunarathne, T., Bae, S., et al. "Twister: a runtime for iterative MapReduce," Proc. ACM Symp. High Performance Distributed Computing (HPDC 10). ACM New York, 2010, pp. 810-818, doi: 10.1145/1851476.1851593.

[10] Eltabakh, M., Özcan, F., Sismanis, Y., Haas, P., Pirahesh, H., et al. "Eagle-eyed elephant: split-oriented indexing in Hadoop," Proc. International Conf. Extending Database Technology (EDBT 13). ACM New York, 2013, pp. 89-100, doi: 10.1145/2452376.2452388

[11] Fruchterman, T., Reingold, E. M. "Graph drawing by force-directed placement," Softw. Pract. Exper. 21, 11 (Nov. 1991), pp. 1129-1164.

[12] Gao X., Roth, E., McKelvey, K., Davis, C., Younge, A., et al. "Supporting a Social Media Observatory with Customizable Index Structures - Architecture and Performance," book chapter to appear in Cloud Computing for Data Intensive Applications, to be published by Springer Publisher, 2014. Available at http://salsaproj.indiana.edu/IndexedHBase/paper_bookChapter.pdf.

[13] Gao, X., Qiu, J. "Social Media Data Analysis with IndexedHBase and Iterative MapReduce," Proc. Workshop on Many-Task Computing on Clouds, Grids, and Supercomputers (MTAGS 2013) at Super Computing 2013. Denver, CO, USA, Nov. 17th, 2013.

[14] Harp project. http://salsaproj.indiana.edu/harp/index.html.

[15] HIndex. https://github.com/Huawei-Hadoop/hindex.

[16] IndexedHBase. http://salsaproj.indiana.edu/IndexedHBase.

[17] McKelvey, K., Menczer, F. "Design and prototyping of a social media observatory," Proc. International Conf. World Wide Web companion (WWW 13). International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland, 2013, pp. 1351-1358.

[18] MongoDB. http://www.mongodb.org/.

[19] Nishimura, S., Das, S., Agrawal, D., Abbadi, A. E. "MD-HBase: A Scalable Multi-dimensional Data Infrastructure for Location Aware Services," Proc. IEEE International Conf. Mobile Data Management (MDM 11). IEEE Computer Society Washington, DC, 2011, pp. 7-16, doi: 10.1109/MDM.2011.41.

[20] Raghavan, U., Albert, R., Kumara, S. "Near linear time algorithm to detect community structures in largescale networks," Physical Review E 76, 036106 (2007).

[21] Riak. http://basho.com/riak/.

[22] Twitter Streaming API. https://dev.twitter.com/docs/streaming-apis.

[23] Vavilapalli, V. K., Murthy, A. C., Douglas, C., Agarwal, Sharad., Konar, M., et al. "Apache Hadoop YARN: yet another resource negotiator," Proc. ACM Symp. Cloud Computing (SoCC 13). ACM New York, 2013, Article No. 5, doi: 10.1145/2523616.2523633.

[24] Weng, L., Flammini, A., Vespignani, A., Menczer, F. "Competition among memes in a world with limited attention," Nature Sci. Rep., (2) 335, 2012.

[25] Zobel, J. Moffat, A. "Inverted files for text search engines," ACM Computing Surveys, 38(2) - 6. ACM New York, 2006.