© 2017 IJSRST | Volume 3 | Issue 3 | Print ISSN: 2395-6011 | Online ISSN: 2395-602X
Themed Section: Science and Technology

Data Mining with Elastic

Mani Nandhini Sri, Mani Nivedhini, Dr. A. Balamurugan
Sri Krishna College of Technology Coimbatore, Tamil Nadu, India

ABSTRACT

Recently, new "big data" technologies and architectures, including Hadoop and NoSQL databases, have evolved to better support the needs of organizations analyzing such data. In particular, Elastic a distributed full-text search engine explicitly addresses issues of scalability, big data search, and performance that relational databases were simply never designed to support. In this paper, we reflect upon our own experience with Elastic and highlight its strengths and weaknesses for performing modern mining software repositories research.
Keywords: Hadoop, NoSQL, Data Mining, Elastic, RESTful API, RDBMS, MySQL

I. INTRODUCTION

The "cloud computing" and "big data" have become less exotic as various search providers aggregate the data from all around the Web. Analysts now understand that they can access whatever information they need almost instantly; in turn, this drives them to search for new tools that can ease their everyday routines as the scale of the tasks at hand broaden and become more ambitious. Most of data analysis process require significant preprocessing of the voluminous raw data: it must be cleansed, filtered, and organized into a usable format for querying. Often, this step is a one-time effort because the goal is to perform an "offline analysis", rather to provide ongoing support for interactive querying of "live" and growing online data. Consequently, researchers often opt to store the processed data in a well known RDBMS, such as MySQL or Oracle, which are simple to set up and easy to query. While this kind of approach is likely the best choice for some tasks, it is not well suited to the needs of Big Data analysis, which requires supporting much larger volumes of data, accommodating the real-time nature of incoming data, and providing quick responses to queries. In a traditional RDBMS, large data requires the creation of many indices to reduce the execution time of queries; at the same time the presence of indices greatly slows the process of updating the data. Since neither of these choices is practical in the presence of Big Data, it is worthwhile to consider what other approaches might work well in this space. In this paper we report on our experience with Elastic, an open source search engine that provides near real-time search and full-text search capability, as well as a RESTful API.

II. METHODS AND MATERIAL

1. ELASTIC

Elastic is an open source full-text search engine written in Java that is designed to be distributive, scalable, and near real-time capable. The Elastic server is easy to install, and the default configuration supplied with the server is called node. sufficient for a standalone use without tweaking, although most users will eventually want to fine tune some of the parameters. A running instance of the Elastic server is called a node, and two or more nodes can form the Elastic cluster. To set up an Elastic cluster, the only value that needs to be set in the configuration file is the name of a cluster; Elastic will take care of discovering nodes on the network and binding them into a cluster.

2. Background

While Elastic is based on Apache Lucene; each Elastic index consists of one or more Lucene indices, called shards. The number of shards that each index has is a fixed value that is defined before the index can be created. When a document is added to an index, the

Elastic server defines the shard that will be responsible for storing and indexing that document. By doing this, Elastic balances the loads between available shards and also improves overall performance, since all shards can be used simultaneously. While such automatic sharding is only one key part of the distributed nature of Elastic, the other part of it is automatic distribution of shards among the nodes in a cluster. For example, imagine we have an index that consists of six shards and that our cluster has only one node. In this case, all six shards will be on the same node; however, if we add one more node to the cluster, Elastic will automatically move half of the shards to this new node, and we will then have two nodes with three shards each. Regardless of the number of shards in an index or the number of nodes they occupy, an index is always seen to a client as a single entity and traditional RDBMSs differ in many ways, at the higher-level many of the core concepts of Elastic have analogues in the RDBMS world (Table 1). All data in Elastic is stored in indices. An index in Elastic is like a database in a RDBMS: it can store different types of documents, update them, and search for them. Each document in Elastic is a JSON object, analogous to a row in a table in a RDBMS. A document consists of zero or more fields, where each field is either a primitive type or a more complex structure. A document has a Document type associated with it; however, all documents in Elastic are schema-free, which means that two documents of the same type can have different sets of fields. Document type here is similar to the RDBMS notion of a table: it defines the set of fields that can be specified for a particular document.

Table 1: Elastic vs. SQL

Elastic element	SQL element
Index	Database
Mapping	Schema
Document type	Table

Elastic is based on Apache Lucene; each Elastic index consists of one or more Lucene indices, called shards. The number of shards that each index has is a fixed value that is defined before the index can be created. When a document is added to an index, the Elastic server defines the shard that will be responsible for storing and indexing that document. By doing this, Elastic balances the loads between available shards and also improves overall performance, since all shards can be used simultaneously. While such automatic sharding is only one key part of the distributed nature of Elastic, the other part of it is automatic distribution of shards among the nodes in a cluster. For example, imagine we have an index that consists of six shards and that our cluster has only one node. In this case, all six shards will be on the same node; however, if we add one more node to the cluster, Elastic will automatically move half of the shards to this new node, and we will then have two nodes with three shards each. Regardless of the number of shards in an index or the number of nodes they occupy, an index is always seen to a client as a single entity.

3. Near Real-Time Search

The search in Elastic is near real-time. It means that although documents are indexed immediately after they are successfully added to an index, they will not appear in the search results until the index is refreshed. The Elastic server does not refresh indices after each update, instead it uses a specified fixed time interval (the default value is 1 second) to perform this operation. Since refreshing is costly in terms of disk I/O, it might affect the speed of adding new documents. Therefore, if you need to perform a large number of updates at once, you might want to temporally increase the default indexing interval value (or even disable auto-refresh) and then manually refresh indices after updates are completed.

4. Performing a Search

Elastic provides its own query language based on JSON called Query DSL. A given search can be performed in Elastic in two ways: in a form of a query or in a form of a filter. The main difference between them is that a query calculates and assigns each returned document with the relevance score, while a filter does not. For this reason, searching via filters is faster than via queries. The official documentation recommends using queries only in two situations: for full text searches or when the relevance of each result in the search is important. For simplicity, we will use term query to describe both queries and filters; however, our experience with Elastic is limited to working only with filters, thus we do not report about use of queries. To execute a search, a client sends a search request to one of the following addresses:

```
http://<server>/_search
http://<server>/_search
http://<server>/_search
{"query": {"filtered": {
"query": {"matchall": {}},
 "filter": {"and": [{
   "range": {
     "modifiedts": {
       "gte": 0, "lt": 1400000000000}}},
{
"term": {
  "reportedby": "XXXX"}},{
  "terms": {
  "bugstatus":
     ["new", "re opened"]}}, {"not": {"term": {"
priority": "p1"}}}]}}}, "from": 0, "size":
100, "fields": ["bugid"]
}
```

Figure 1: A sample search query using filters.

The first URL represents a search on all indices on the server, while the last one searches only the documents of a particular type withing a particular index. An example of the search query using filters is shown in Figure 1. An Elastic query is broadly similar to a SELECT query in SQL. The filter field specifies the conditions that must be met to return a document, similar to the WHERE clause in a SQL query. Unlike SELECT in SQL, where one must specify the list of tables to be joined before filter conditions are applied, in Elastic the scope of the search is restricted by the URL the query is sent to, and queries are likely to differ only in their filter conditions. Thus, such a query can serve as a boilerplate for a variety of further queries: one need only put the required conditions into the filter clause, and a new query is ready to be executed. The fields from and size are used for pagination (similar to LIMIT clause in MySQL) — the former sets the first document to be returned, while the latter sets the maximum number of documents in the returned set. The field fields allows the selection of specific fields of interest in the document to be returned. If no fields are listed, the query performs similar to SELECT * FROM and results in all fields of the document being returned. As the data is sent from the server over HTTP in a text form, reducing the number of fields will likely improve overall performance.

5. Strengths

Scalability. Before researchers can do anything with Elastic, they need to decide where they want their data to be stored. A relational database can be used for this; however, all the data is typically stored in a single database. As a result, the more data we store, the more powerful server we need (vertical scaling). The server can be pushed to its limits very quickly, and we would need to start sharding our database and putting each shard on different servers (horizontal scaling). Unfortunately, it is not a trivial operation and to the best of our knowledge none of the current relational database provides this functionality outof-box. Elastic automatically distributes shards of an index across the nodes of a cluster and controls that they are loaded equally. So if you expect to add more data in future and want to accommodate the growth in data, Elastic is your choice as it scales horizontally. Agility. Data can be agile in terms of the number of updates or new records, in terms of constantly changing structure of a logical piece of information or document, or a combination of both of them. Relational databases are good at changing/adding data as long as the amount of data in a database is not too large. The time needed to perform a database maintenance (mainly recalculating indices) increases with its size. Elastic can better handle agile data because of a) each of the shards is being indexed/refreshed independently, and b) indices are constantly refreshed with fixed time interval, which means that it is unlikely that a shard has accumulated a lot of unrefreshed data. In the RDBMS world, a database schema is fixed and known before the first record arrives. If any updates might take place, the schema must be changed and this must be propagated to the records that are already in the database. If the database stores big data, this process can be very slow. Additionally, if the database is used in a domain where documents can have a lot of optional fields, the database can end up having large sparse tables that waste disk space for storing NULLs. Elastic does not impose schema on the documents in indices. If a new document is added to an index and there is a new field in this document, Elastic will automatically update the mapping. There is no need to change already stored documents since they do not have such a field. In addition, Elastic can automatically alter the data type of a field if a value in a new document requires a "wider" type (e.g., changing integer to long).

Performance. Best practises of the relational databases world dictate that each relational database must go through the normalization process during the design phase. By converting a database to a particular normal form we decouple bulk data into several tables and minimize the amount of redundant information. While the normalization benefits the create, update, and remove operations, it is likely to complicate the read operations. Most SELECT statements hit more than one table and they must be joined before the filtering conditions are applied. Although every RDBMS handles this operation as efficiently as it can, it is a time-consuming process if the query involves complex schemas. But since Elastic is document-oriented it does not need to spent time on this preliminary step (i.e., gathering the data). Moreover, all shards within an index are searching for the documents satisfying some filter criteria concurrently, and after that results from all them are combined and returned. While scalability and schema-free documents are common for NoSQL systems, the combination of all three (scalability, agility, and performance) in one system is what makes Elastic stand out from other systems.

6. Weaknesses

Learning curve. The JSON origin of the Elastic query language makes it really easy to start writing simple queries. However, query writing becomes more complicated if it involves nested objects. There is a special type of queries, nested queries, in Elastic that must be used when one of the filter conditions is a condition on a field from a nested object. The use of such queries requires the understanding of how particular documents are stored and analyzed by Elastic (i.e., the mapping that is currently in use).

Finally, Elastic inherits some weaknesses of being a NoSQL system — lack of transactions, lack of JOIN operation, possible inconsistencies in data, etc.

III. RESULTS AND DISCUSSION

APPLICATIONS

In this section we provide a concrete example of the system that uses Elastic. We also speculate about the boundaries of a domain where Elastic should be chosen over other systems.

7.1 Software Analytics

Elastic is best suited for the applications that are built to handle real time data that needs to be processed and analyzed in a rapid manner. Such applications include software analytics. As an example of software analytics applications, “Captome” implements the Elastic to do its quantitative and intutive analysis search.

CaptoMe is a comprehensive NextGen Research Discovery platform that enables researchers, consultants, clinical trial planners and biomedical professionals to easily combine biomedical content with a set of tools for information management and results reporting.

Captome created two indices on their own servers: one for research documents and one for Clinical trials documents. Each index contained 50 millions of documents.

Moving to the Elastic server and being able to execute real time search on the issue repository improved the performance of our tool being very crucial for Captome to be useful to the researchers in biomedical companies. We noticed a big improvement in the execution time, the average response time of querying and displaying the information on the web site was reduced from 30 seconds (using MySql servers) to 1.5 seconds (on Elastic servers). The response time of a larger is now 2.5 seconds compared to the previous 2.5 minutes.

7. Other Applications

Elastic is a good choice if you expect to perform any real-time analysis of the data such as as social media analysis, or if you need to provide real-time search that scales to terabytes of information. Thousands organizations worldwide including Netflix, Facebook, GitHub and Stack Overflow, have adopted Elastic to help them overcome limitations of their old approaches in handling new demands of agile data processing and storage. For example, for McGraw-Hill Education Labs, Elastic has provided scalability and high performance to their personalized student learning system. For Xing, Europe’s largest professional social network, Elastic brings the ability to handle one million updates per day

in real-time and search performance to support 6 million queries per day. With Elastic we can answer a wide range of questions, from the aggregated information to a fine grained data pieces. We believe that in time current MSR like techniques, in particular, creating traditional offline databases, loading data and use database querying tools, will be replaced by setting up Elastic clusters, writing and performing ETL (execute, transform and load) jobs and performing real-time large data analysis using Elastic search functionality. While Elastic is currently adopted by industrial organizations (mainly due to high costs of hardware and experts), research community is also evolving and exploring solutions such as Elastic or Hadoop to be able to mine modern data repositories.

IV. CONCLUSION

Elastic is an easily scalable, full-text search engine that is capable of handling large amounts of online and schema-less data. We built a tool called Captome that allows researchers, consultants, clinical trial planners and biomedical professionals to easily search and analysis process. By switching to Elastic, Captome remarkably improved their performance which rendered the tool suitable for the real-time use.

V. REFERENCES

[1]. Baysal, R. Holmes, and M. W. Godfrey. Developer dashboards: The need for qualitative analytics. IEEE Software, 30(4):46–52, 2013.

[2]. Sematext. Elastic refresh interval vs indexing performance. http://bit.ly/1iZoPGc, July 2013.

[3]. Hao Zhang , Gang Chen"In Memory Pattern Mining Data"in IEEE Transactions on Knowledge and Data Engineering Volume: 27, Issue: 7, July 1 2015.

[4]. Ge song, Justine Rochas"K Nearest Neighbour Joins For Bigdata On Map Reduce" in IEEE Transactions on Knowledge and Data Engineering Volume: 28, Issue:9, Sep 1 2016.

[5]. Ling Chen ,Xue LI "Mining Health Examination Records-A Graph Based Approach" in IEEE Transactions on Knowledge and Data Engineering Volume: 28, Issue:9, Sep 1 2016.