Adaptation of Apriori to MapReduce to Build a Warehouse of Relations Between Named Entities Across the Web

Jean-Daniel Cryans
Software engineering and IT Dept
École de technologie supérieure
Montreal, QC, Canada
Email: jdcryans@gmail.com

Sylvie Ratté
Software engineering and IT Dept
École de technologie supérieure
Montreal, QC, Canada
Email: Sylvie.Ratte@etsmtl.ca

Roger Champagne
Software engineering and IT Dept
École de technologie supérieure
Montreal, QC, Canada
Email: Roger.Champagne@etsmtl.ca

***Abstract*—The Semantic Web has made possible the use of the Internet to extract useful content, a task that could necessitate an infrastructure across the Web. With Hadoop, a free implementation of the MapReduce programming paradigm created by Google, we can treat these data reliably over hundreds of servers. This article describes how the Apriori algorithm was adapted to MapReduce in the search for relations between entities to deal with thousands of Web pages coming from RSS feeds daily. First, every feed is looked up five times per day and each entry is registered in a database with MapReduce. Second, the entries are read and their content sent to the Web service OpenCalais for the detection of named entities. For each Web page, the set of all itemsets found is generated and stored in the database. Third, all generated sets, from first to last, are counted and their support is registered. Finally, various analytical tasks are executed to present the relationships found. Our tests show that the third step, executed over 3,000,000 sets, was 4.5 times faster using five servers than using a single machine. This approach allows us to easily and automatically distribute treatments on as many machines as are available, and be able to process datasets that one server, even a very powerful one, would not be able to manage alone. Based on these findings, we can generalize that, with this great scalability, processing more data would be faster, depending on the number of servers used.**

***Keywords*-web mining; association mining; Apriori algorithm; MapReduce paradigm;**

I. Introduction

Most of the data mining algorithms written during the last decade have been designed to execute on a single computer. In the Semantic Web era, it is becoming increasingly easy to extract information from the Web, provided that we can store these data which no longer reside on a single machine. We must therefore find ways to parallelize the processing on several machines, in order to increase both storage capacity and treatment. This study proposes an adaptation of the search algorithm Apriori to the MapReduce programming paradigm to deal with news feeds to find the relations between named entities. This new algorithm, called AprioriMR, will use the knowledge contained on the Internet to reveal links between cities, people, objects, monuments, or other tangible concepts.

The rest of the article is organized as follows. Section II explains the background of this research. Section III describes the objectives of the adaptation. Section IV presents the tools used to obtain the data and process them. Section V explains how data are processed and how the Apriori algorithm was adapted. Section VI presents the results of the study. Section 7 puts this study into perspective with respect to other, similar proposals.

II. Background

This study is part of the Semantic Web effort to make information available to users and machines. In it, we seek to more precisely extract the relationship between named entities in news feeds. The news comes from companies known for the quality of their content, such as newspapers and television stations that publish text online. This is to ensure: 1) that the news can be interpreted in a programmatic way; and 2) that the news source is valid, and therefore that the information published is correct. These properties are discussed in more detail in Section IV.

As it is currently impossible to make useful queries on the content stored only on the Web, we must do this on our own infrastructure. The data for large numbers of pages cannot be contained on a single machine, because their volume easily exceeds the size of the largest hard disks. Even if this were possible, the loss of that single machine would make the content unavailable for a time. Furthermore, the serial processing of these data would be limited by the speed of the computer components. This is why software such as Hadoop [1], which facilitates this type of processing, have been designed. Hadoop provides a distributed file system that can be used on more than a thousand machines and an implementation of the MapReduce programming paradigm [2], invented by Google, which allows data to be processed reliably on these machines. XXX-COMMENT–*end the Section II with an evaluation on 'why' [1] and [2] are not enough; what is essentially missing and your work fills in*–COMMENT- IDEE-SR: At the time of writing no implementation of Apriori within Hadoop exists. While the

itemset counting is trivial in Map/Reduce, the real challenge of the algorithm is in the candidate selection implementation. XXX

At the end of the study, the system created should be able to take a named entity and find other entities that are very closely linked to it on the basis of Web content. This will necessarily lead to the disambiguation of entities (the word apple, for example, can refer to a fruit, a computer company, or a multimedia corporation). From there, it will also be possible to determine the strength of the relationships between several entities.

III. Objectives

This study has three objectives: 1) to adapt the Apriori learning algorithm of association rules [3] to the MapReduce paradigm [4]; 2) to apply the OpenCalais Web service [9] to a large number of articles from the Internet in a distributed manner; and 3) to evaluate the database HBASE [8] in a batch processing environment, as a warehouse capable of responding to requests in real time.

Adapting the Apriori algorithm to MapReduce is necessary to deal with sets of transactions that cannot be stored in RAM on a single computer. This adjustment will help treat millions of items from the Web several times a day and obtain results equivalent to those of the original algorithm. Since Apriori is not designed to run on a cluster of machines, we must first demonstrate whether or not this is feasible. It should also be mentioned that the MapReduce framework imposes a minimum cost in terms of time for each execution. Consequently, we must determine the performance loss caused by the use of MapReduce on data that can be processed on a single machine.

Furthermore, the feasibility of using OpenCalais for very large sets of articles must be shown, because this is the tool used to find the named entities. We must then be able to send tens of thousands of articles per day to this Web service and not be constrained by the response time, which is more than one second. In other words, a method must be properly designed to use OpenCalais in a distributed manner.

HBASE is a database inspired largely by Bigtable [10], which allows Google to store terabytes of structured data in a scalable way. However, it does not offer a means of interaction with the SQL database language, as it is not relational, but it is nevertheless compatible with MapReduce. We must then validate that HBASE will be able to store millions of articles from the Web along with the itemsets that will be extracted. In addition, we must ensure that we can quickly retrieve specific information despite the lack of relationships between tables, in order to serve a website and deal with batch processing simultaneously.

IV. Data and Software

The first source of data for this study was the thousands of RSS feeds of newspapers and television stations which publish articles online. Each RSS feed contains a series of recent articles that are described by title, abstract, first lines of the text, date, and, sometimes, attached files (images, etc.). Each item in an RSS feed then refers directly or indirectly to an HTML page containing an article described by a title, subtitle, and text, but also by general elements of each site (menus, advertising, and links to related articles).

```
<rdf:Description rdf:about=
  "http://d.opencalais.com/comphash-1/
  64136b2b-cb4e-36ac-9f32-f58f4c1f1c8a">
 <rdf:type rdf:resource=
    "http://sopencalais.com/1/type/
        em/e/Company" />
  <c:name>British Airways </c:name>
  <c:nationality>British</c:nationality>
</rdf:Description>
```

Figure 1. Example of a resource in RDF format

The use of RSS feeds has several advantages. First, the use of a Web crawler, which is often difficult to configure, is no longer necessary, since it only takes one library to read the RSS and another to give the HTML of a URL. Also, it helps to have a high level of control over which pages will be covered, because, without a proper and expensive configuration, crawlers such as Nutch [11] and Heritrix [12], have to analyze both images and link forms, recovering a large number of pages in the process that have no value in the search for well-written articles and valid content.

The second data source is OpenCalais, a Web service which makes it possible to find named entities in a text. The result is presented by default as a Resource Description Framework (RDF) developed by the W3C to describe resources that come from the Web.

For example, the resource

```
http://d.opencalais.com/er/geo/city/
ralg-geo1/b3719a18-c511-51a8-b3f9-
f8480b3b6e48.html
```

is a reference to the City of New York in the United States.

An RDF response contains many resources, as shown in Figure 1, where British Airways has a URI, a name, a type, and a nationality. It attributes change depending on the type of response.

Using OpenCalais allows us to respond easily to the problem of recognition of named entities, but there are several constraints. First, articles sent to the service cannot have more than 100,000 characters, otherwise an error is returned. Second, a maximum of 40,000 queries per day and four requests per second can be sent to the service. Third, the submitted texts must be very well built (title, subtitle, and text) so that the service is able to detect entities. This is because the emphasis is on accuracy and not on recall.

Despite these limitations, it is much more economical to use OpenCalais than to develop our own library for named entity detection.

The choice of Apache Hadoop (which contains an implementation of MapReduce and a distributed file system) and HBASE is justified by the high level of scalability and availability they offer. It thus becomes possible to store several terabytes of data and process them easily and reliably, since the software stack is resistant to engine failures and mistakes. The versions we used were Hadoop 0.19.1 and HBASE 0.19.3.

A cluster of ten computers was used to store data with Hadoop and HBASE. The master node was composed of a processor running at 1.8 GHz, with 2 GB of RAM and two disks as mirrors. The other nine slave nodes had the same processor and 1 GB of memory, with two 80 GB disks used independently by the distributed file system, for a total of approximately 1 TO. A cluster of five computers (each with a 2.4 GHz processor and 1 GB of memory) was used to process the data with MapReduce. The master process of MapReduce was on the same master node of the first cluster. Such a configuration is not consistent with MapReduces principle of proximity, because normally all slave processes are found on all slave nodes. But, with one processor and little memory, it is not possible to combine the processes without competing for resources.

V. Methods

Thanks to these tools and data sources, it is possible to acquire data, as shown in Figure 2. Several times a day, the following operations are performed in sequence.

1) First, a set of RSS feeds is analyzed to obtain the URLs of new articles that will be saved in HBASE with a label indicating that they are new and should therefore be treated (see 1a and 1b in Figure 2).
2) The second task, associated with the Map phase, consists of going through the HBASE table of all articles to download the HTMLs of the new articles. It then communicates with OpenCalais to get the RDF, filter entities with a relevance index of 2 (which is too low), and generate the set of all subsets of the remaining entities (except the empty set), known as the Powerset, which will be forwarded to the Reduce phase. This last phase will record this new set of itemsets in the table of articles and mark the page as read. As the task is going through the full table of items, it also notes the final number of articles with itemsets, which will become the total number of transactions (see 1a, 2b and 2c in Figure 2).
   In order not to limit the reading speed of pages to the number of servers (in this case, five), each Map reads three pages at a time. After each page is read, there is a down time of five seconds, as a courtesy to the site analyzed. Moreover, since OpenCalais limits the number of articles processed per second to four, it is likely that one or more processes will receive an error from the Web service. When this occurs, a random waiting time of between 0.1 and 10 seconds is observed before trying again, so as not to reproduce the same problem indefinitely.
3) The third task in the Map phase reads the table of articles and forwards each of the itemsets individually to the Reduce phase, which will in turn obtain the number of occurrences of each of the itemsets. Only itemsets will be saved, the number of which, divided by the total number of transactions, called support, is greater than a predefined metric set when configuring the task (see 3a and 3b in Figure 2).
4) Finally, several other tasks are performed to obtain aggregates (see, in particular, Section VI), or to index the data for a search engine (see 4 in Figure 2).

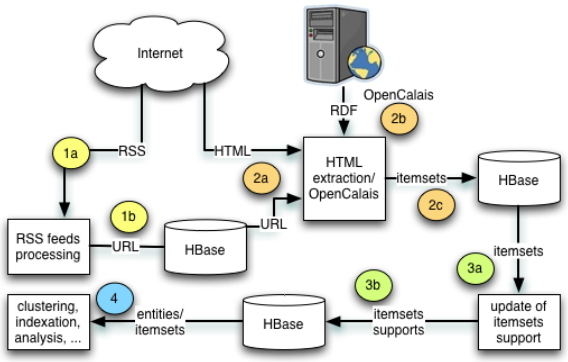

Figure 2. Process Data Acquisition Process

COMMENT– –COMMENT–RESPONSE? The pseudo-code of our adaptation is presented below.

VI. Results

The data acquisition task has been performed regularly for a month on a growing number of RSS feeds, and has produced more and more data every day. At the time of writing, about 1,500 feeds are cycled every day, which helped to analyze more than 100,000 web pages. Approximately 20% of the pages covered were either unavailable (HTTP 400 or 503 errors, for example) or available, but too small or too big, or their encoding was not well specified. From these pages, over 50,000 named entities were discovered and the number of itemsets not filtered reached more than 3 million. The disk space required to contain these data is approximately 30 GB uncompressed, but replicated three times, so the total space required is about 90 GB. Two data computers failed during testing, but this had no impact on the results.

The adaptation of the Apriori algorithm to MapReduce to make the processing task extensible was verified by running the third step of the method on a different number of

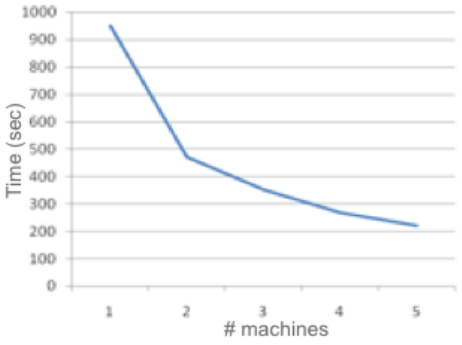

Figure 3. Execution time (sec) of stage 3, according to the number of machines

machines, from one to five. The number of machines in the data cluster was not changed, as this would have influenced the maximum flow. Between each run, HBASE was rebooted to clear statements in memory and in order to benefit from the same optimizations of the Sun Java Virtual Machine each time. The Map number was established at 40 for all treatments and the Reduce number was equivalent to the number of machines. The results are shown in Figure 3. We can see that moving from one to five machines reduces the execution time by a factor of 4.5. This is because each Map can draw information from one machine at a time, which leaves the rest of the cluster data dormant.

The reason why the execution with five machines is not exactly five times faster is the quality of the data distribution in HBASE: it can easily happen that two Maps are using the same data machine at the same time, inducing a slowdown. The Reduce phase with a single machine is also faster, because it resides on a virtual machine and discovers the location of the data only once. When using five machines, each of them must find where to place the data in HBASE.

The experiment demonstrates that it is indeed possible to use OpenCalais in a Web-crawling task, but we must be able to handle the errors returned. As described in Section V, it is useless to resend our requests directly to the Web service immediately, or to wait for a fixed period of time prior to doing so. It is best to choose a random time between shipments. Another issue is the occurrence of unexpected errors. Sometimes, for example, after several attempts, the service does not respond within the maximum period of 15 seconds, while other pages are handled at the same time without a problem on other servers. In addition to the random wait, experience shows that we must also perform a maximum number of sends and then give up, in order not to lose too much time on the treatment of a single page.

The final validation was to ensure that HBASE was able to contain millions of elements in addition to being able to respond to requests despite the lack of a relationship between tables and a different data model, while dealing with batch processing. With millions of elements, we tried to filter none of the itemset at the third stage, and therefore to write the 3 million items in an HBASE table. While the treatment was successful, it took much longer, and had the disadvantage of slowing down the analytical tasks that followed. The first task to run at that point was the detection of the itemsets with the largest supports for each cardinality (from one to twelve). To do this, the Map phase would cover the itemset table in full. Its execution time is approximately 25% higher for about 130 times more elements, which is normal, given that scanning an HBASE table usually takes less than a millisecond per line. Despite these additional data, the website developed in [7] is not slower, since each request, which is performed by a row key and a column key, is very fast, no matter how many rows are involved (one to a billion), because of its distributed model maps.

VII. Discussion

Our work is similar to that of [5], whose algorithm processes uncertain data flows. Flows are characterized by a number of transactions containing items, and must be treated immediately because they are no longer available for reuse. In addition, items are not distributed evenly in each stream, which means that an item that is uncommon at one moment may become very common later on. Uncertain data are characterized by an existential uncertainty that indicates the degree of certainty that each item will be in each transaction, which is not the case for typical transactional databases where this uncertainty does not exist. Their flows for data distribution resembles in a way to our RSS stream, since it is uneven, the difference being that we keep the stream available so that it is always possible to process it again. Since it is not necessary to provide a limited-stream window to users, all the available information is used. The existential uncertainty is almost identical to the degree of relevance provided by OpenCalais: if an entity is not relevant to the article, it is probably part of the noise that surrounds it. At this time, the support calculation does not take into account the relevance of the entities, because we think that, beyond a certain level, confidence is sufficient to include the entity. In addition, levels of support are so low that filtering them with the help of this probability would only reduce their number still more.

COMMENT–*It should be considered to mention relationship to other studies investigating distributed approach to Apriori algorithm. Cristian Aflori and Mitica Craus: Grid implementation of the Apriori algorithm, Advances in Engineering Software, Volume 38, Issue 5, May 2007, Pages 295-300*–COMMENT

The work of Dean and Ghemawat [4] on MapReduce at Google shows that a software framework for distributed processing over thousands of machines is feasible and can be reliable. The paradigm allows for parallel treatment of different domains without resorting to complicated logic.

The Map phase of each task takes as input a set of keys and values on which treatment will be performed to create other keys and other values. When this phase is completed, all values are grouped according to their key. These groupings are then forwarded to the Reduce phase, which performs further processing to create other keys and values. Each Map/Reduce phase is separated into sub-tasks distributed on the machines of the cluster, and, if one of them is missing, its sub-tasks are automatically redirected to other machines in a transparent manner. It is also possible to make further adjustments to MapReduce, such as writing directly into a compatible database (Bigtable at Google, or HBASE in Open Source).

The use of this paradigm can easily make a serialized processing task extensible. Since the Map/Reduce writing tasks are not dependent on the number of machines, the same code can be used on five or a thousand machines. In order to process documents on the Web, it is very advantageous to adapt Apriori. The tasks described in this study are consistent with the model described in [4]. The distribution potential is used to read multiple feeds simultaneously on multiple machines.

VIII. Conclusion

Adapting the Apriori algorithm to MapReduce is a valid solution to the problem posed by the amount of data to be processed on the Web. The use of a cluster of machines to distribute the treatment has helped to optimize RSS stream-mining, and allows the complete processing of multiple streams (reading the articles and sending them to OpenCalais). MapReduce also achieved very rapid association-mining between named entities in a sample of 3 million itemsets stored in HBASE, and could handle even more data. Moreover, if, at some point, the treatments become too long, it will be possible to add new resources without modifying the algorithm. This study serves as the basis for the dynamic website developed in [7], which takes the processed data and presents it to users. Future work could focus on cleaning up Web pages to remove advertising and static content (such as a menus) to improve the detection of entities. A window like the one proposed by Leung and Hao [5] could also be used to present only current and up-to-date information.

References

[1] A. Bialecki, A. Murthy, C. Douglas, D. Cutting, D. Das, D. Borthakur, E. Soztutar, A. Gates, J. Kellerman, M. Konar, N. Daley, O. Natkovich, O. O'Malley, P. Hunt, M. Stack, C. Taton, and T. White. (2009) Hadoop core. [Online]. Available: "http://hadoop.apache.org/core/", 2010/15/01.

[2] H.C. Yang, A. Dasdan, R.-L. Hsiao, and D. Parker, "Map-reduce-merge: simplified relational data processing on large clusters," in *ACM SIGMOD International Conference on Management of Data*, 2007, pp. 1029 – 1040.

[3] R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," in *ACM SIGMOD International Conference on Management of Data*, 1993, pp. 207 – 216.

[4] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, 2008, pp. 107–113.

[5] C. Leung, and B. Hao, "Mining of Frequent Itemsets from Streams of Uncertain Data," in *IEEE International Conference on Data Engineering*, 2009, pp. 1663–1670.

[6] C. Aflori and M. Craus. "Grid implementation of the Apriori algorithm," in *Advances in Engineering Software*, Vol 38, 2007, pp. 295–300.

[7] J.-D. Cryans, "Prototypage d'un engin de recherche de relations entre des entités nommées," *Projet de fin d'études*, École de technologie supérieure, 2009, pp. 1663–1670.

[8] N. Joffe, J.-D. Cryans, B. Duxbury, J. Kellerman, A. Purtell, M. Stack, and R. Rawson. (2009) Hbase. [Online]. Available: "http://hbase.org", 2010/15/01.

[9] OpenCalais. (2009) OpenCalais Web Service. [Online]. Available: "http://www.opencalais.com", 2010/15/01.

[10] F. Chang, J. Dean, S. Ghemawat, W. Hsieh, D. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. Gruber, "Bigtable: A distributed storage system for structured data," *ACM Transactions on Computer Systems (TOCS)*, vol. 26, no. 2, 2008, Article 4.

[11] A. Bialecki, M. Cafarella, J. Charron, D. Cutting, O. Gospodnetić, D. Güney, P. Kosiorowski, D. Kubes, C.A. Mattmann, S. Siren, and J. Xing. (2009) Lucene Nutch. [Online]. Available: "http://lucene.apache.org/nutch/", 2010/15/01.

[12] Multiple authors (2009) Heritrix crawler. [Online]. Available: "http://crawler.archive.org/", 2010/15/01.