CONTENT-BASED RECOMMENDATION ALGORITHMS ON THE HADOOP MAPREDUCE FRAMEWORK

Toon De Pessemier, Kris Vanhecke, Simon Dooms and Luc Martens
IBBT, Ghent University, Gaston Crommenlaan 8 box 201, Ghent, Belgium

Keywords: Recommender systems, Cloud computing, Hadoop, MapReduce, Content-based recommendations.

Abstract: Content-based recommender systems are widely used to generate personal suggestions for content items based on their metadata description. However, due to the required (text) processing of these metadata, the computational complexity of the recommendation algorithms is high, which hampers their application in large-scale. This computational load reinforces the necessity of a reliable, scalable and distributed processing platform for calculating recommendations. Hadoop is such a platform that supports data-intensive distributed applications based on map and reduce tasks. Therefore, we investigated how Hadoop can be utilized as a cloud computing platform to solve the scalability problem of content-based recommendation algorithms. The various MapReduce operations, necessary for keyword extraction and generating content-based suggestions for the end-user, are elucidated in this paper. Experimental results on Wikipedia articles prove the appropriateness of Hadoop as an efficient and scalable platform for computing content-based recommendations.

1 INTRODUCTION

Content-based (CB) recommendation techniques are based on content analysis, usually through metadata or textual descriptions of the content items previously consumed by the user (Mladenic, 1999). These content items might be annotated by the content authors with characteristic attributes to ease the content retrieval and recommendation process. In the alternative, CB recommender systems have to rely on keyword extraction techniques to obtain characteristic properties from the textual description of the item. These characteristic attributes are then utilized to build a model or profile of user interests. The attributes of the content items consumed by an individual user, together with the associated feedback behaviour (i.e. explicit feedback such as star-ratings or implicit feedback such as reading times) make up the profile of that user. Although the details of various systems differ, generating CB recommendations share in common a need for matching up the attributes of this user profile against the attributes of the content items. Finally, the personal suggestions consist of the content items which are most similar to the content the user consumed and appreciated in the past. CB recommendation techniques have been applied in various domains, such as email , news , and web search. However, the computational complexity of the text processing and profile-item matching is high, which hampers the application of CB recommendation algorithms in large-scale. Tailored implementations can be designed for specific parallel processing architectures, but the Hadoop framework offers a standardized solution for data processing on large clusters (Dean and Ghemawat, 2008).

Previous research on the Hadoop framework proves its scalability (Brown, 2009) and appropriateness for document processing (Elsayed et al., 2008). Nevertheless, this paper is the first to our knowledge to provide details about calculating CB recommendations and pairwise similarities on the framework. Moreover, we investigated the calculation times of the various jobs, needed to generate these recommendations. The remainder of this paper is organized as follows: Section 2 provides a short introduction to the Hadoop framework. Section 3 elaborates on how relevant keywords can be extracted from content descriptions using MapReduce operations. Generating CB recommendations by matching the user profiles and content descriptions is described in Section 4. Section 5 provides some first benchmark results, based on Wikipedia articles, to investigate the required calculation time. Finally, we offer a brief conclusion and point out interesting future work in Section 6.

De Pessemier T., Vanhecke K., Dooms S. and Martens L..
CONTENT-BASED RECOMMENDATION ALGORITHMS ON THE HADOOP MAPREDUCE FRAMEWORK.
DOI: 10.5220/0003193802370240
In *Proceedings of the 7th International Conference on Web Information Systems and Technologies* (WEBIST-2011), pages 237-240
ISBN: 978-989-8425-51-5

2 HADOOP FRAMEWORK

Hadoop MapReduce is a programming model and software framework that supports data-intensive distributed applications. This Apache project is an open-source framework for reliable, scalable, distributed computing and data storage. It can rapidly process vast amounts of data in parallel on large clusters of computer nodes. Hadoop MapReduce was inspired by Google's MapReduce (Lämmel, 2007) and Google File System (GFS) (Ghemawat et al., 2003) papers. MapReduce is based on the observation that many tasks have the same structure: a large number of records (e.g., documents or database records) is sequentially processed, generating partial results which are then aggregated to obtain the final outcome. Of course, the per-record computation and aggregation vary by task, but the fundamental structure remains the same (Elsayed et al., 2008). MapReduce provides an abstraction layer which simplifies the development of these data-intensive applications by defining a *map* and *reduce* operation with the following signature:

$$map : (k_x, v_x) \rightarrow [k_y, v_y] \tag{1}$$

The map operation is applied to every input record, which has the data structure of a key-value pair. This mapper generates an arbitrary number of key-value pairs as an intermediate result (indicated in equation 1 by the square brackets). Afterwards, these intermediate results are grouped based on their key. The reducer gets all values associated with the same intermediate key as an input and generates an arbitrary number of key-value pairs.

$$reduce : (k_y, [v_y]) \rightarrow [k_z, v_z] \tag{2}$$

3 CONTENT CHARACTERIZATION

Many CB recommendation algorithms are based on relevant semantic metadata describing the content items of the system. However, many online systems do not dispose of structured metadata, forcing them to rely on textual descriptions of the content. Therefore, the proposed MapReduce operations, used for calculating item similarities or recommendations, are only dependent on such a set of textual documents describing the content items of the system. To handle these content descriptions, the documents are transformed to characterizing terms and a vector of term weights w_t, which indicate the relevance of each term t for the item.

To identify these terms t and calculate the term weights w_t, we adopted the Term Frequency - Inverse Document Frequency (TFIDF) (Salton and McGill, 1983) weighting scheme. Although the ordering of terms (i.e. phrases) is ignored in this model, it has proved its efficiency in the context of information retrieval and text mining (Elsayed et al., 2008). The TFIDF can be obtained by calculating the frequency of each word in each document and the frequency of each word in the document corpus. The frequency of a word in a document is defined as the ratio of the number of times the word appears in the document, n, and the total number of words in the document, N. The frequency of a word in the document corpus stands for the ratio of the number of documents that contain the word, m, and the total number of documents in the corpus, D.

To calculate the term weights w_t of an item description as TFIDF, the following four MapReduce jobs are executed. The first job calculates the number of times each word appears in a description, n. Therefore, the map operation of this job takes the item identifier (i.e. id) as input key and the content of the description as input value. For every word in the description, a new key-value pair is produced as output: the key consists of the combination of the word and the item identifier; the value is just 1. Afterwards, a reducer counts the number of appearances of each word in a description by adding the values for each word-id combination.

$$\begin{aligned} map &: (id, content) \rightarrow [(word, id), 1] \\ reduce &: ((word, id), [1]) \rightarrow ((word, id), n) \end{aligned} \tag{3}$$

The mapper of the second job merely rearranges the data of the records by moving the *word* from the key to the value. In this way, the following reducer is able to count the number of words in each document, i.e. N.

$$\begin{aligned} map &: ((word, id), n) \rightarrow (id, (word, n)) \\ reduce &: (id, [word, n]) \rightarrow [(word, id), (n, N)] \end{aligned} \tag{4}$$

The third job calculates the number of item descriptions in the corpus that contain a particular word. The mapper of this job rearranges the data and the reducer outputs the number of descriptions containing the word, i.e. m.

$$\begin{aligned} map &: ((word, id), (n, N)) \rightarrow (word, (id, n, N, 1)) \\ reduce &: (word, [id, n, N, 1]) \rightarrow [(word, id), (n, N, m)] \end{aligned} \tag{5}$$

The fourth job, which only consists of a mapper (i.e. the reducer is the identity operation), produces the TFIDF of each id-word pair. The total number of item descriptions in the document corpus is calculated in the file system and provided as an input variable of this MapReduce job. Although it is possible to merge

this last job with job 3, saving one disk IO cycle, we omitted this optimisation in this paper for clarity.

$$map : ((word,id),(n,N,m)) \rightarrow ((word,id),tfidf) \tag{6}$$

4 PROFILE MATCHING

CB recommendation algorithms evaluate the appropriateness of a content item by matching up the attributes of the content description against the user profile. If the user profile and content description are characterised by a vector of term weights, this matching process can be performed by a similarity measure. One of the most commonly used similarity measures is the cosine similarity (Salton and McGill, 1983), which calculates the cosine of the angle between the two vectors. We adopted this similarity measure in this research because of its simplicity and efficiency for matching vectors. To calculate the cosine similarity between a user profile and an item description using MapReduce operations, three jobs are required. The first job calculates the Euclidean norm of each vector. The mapper of this job rearranges the data of the vectors representing the user profiles and item descriptions. Next, the reducer calculates the norm of the vector and appends the result to the key of the records. These operations are performed on the item descriptions as well as the user profiles, i.e. *id* stands for an item or user identifier.

$$\begin{aligned} map &: ((word,id),tfidf) \rightarrow (id,(word,tfidf)) \\ reduce &: (id,[word,tfidf]) \rightarrow [(id,norm),(word,tfidf)] \end{aligned} \tag{7}$$

The second job identifies the item-user pairs, in which the item description and the user profile have at least 1 word in common. Again, the mapper just rearranges the current data records so that the reducer receives the records ordered by word. Next, every item-user pair, in which the item description as well as the user profile contains the current word, are identified and returned as output. The reducer may distinguish the items from the users, based on the identifier or another discriminating attribute of the record.

$$\begin{aligned} map &: ((id,norm),(word,tfidf)) \\ &\rightarrow (word,(id,norm,tfidf)) \\ reduce &: (word,[id,norm,tfidf]) \\ &\rightarrow [word,(id_i,norm_i,tfidf_i,id_u,norm_u,tfidf_u)] \end{aligned} \tag{8}$$

Finally, the third job calculates the cosine similarity of every item-user pair, in which the item description and the user profile have at least 1 word in common. The mapper orders the data records by the identified pairs. Afterwards, the reducer calculates and outputs the cosine similarity for each of these pairs based on the previously calculated intermediate results.

$$\begin{aligned} map &: (word,(id_i,norm_i,tfidf_i,id_u,norm_u,tfidf_u)) \\ &\rightarrow ((id_i,id_u),(word,norm_i,tfidf_i,norm_u,tfidf_u)) \\ reduce &: ((id_i,id_u),[word,norm_i,tfidf_i,norm_u,tfidf_u]) \\ &\rightarrow ((id_i,id_u),sim) \end{aligned} \tag{9}$$

This way, the top-N recommendations can be generated for every user by selecting the N items which have the highest cosine similarity with the user's profile vector. Moreover, the same MapReduce operations can be employed to calculate pairwise similarities. By calculating the cosine similarity between every pair of item vectors, related content items can be identified. Like-minded users can be discovered by executing the MapReduce jobs on the user profiles, i.e. calculating the most similar users based on the cosine similarity of their personal profile vectors.

5 RESULTS

To benchmark the performance of the Hadoop MapReduce framework and test its suitability for calculating CB recommendations, we performed an experiment based on a varying number of input files. Since we had no data of user profiles at our disposal, the scenario of a pairwise item comparison, as described at the end of Section 4, was evaluated. This means the framework had to calculate the similarities of every unique item-item pair that can be composed from the input files; whereas a CB recommendation algorithm compares every item-user pair in the system. Because this pairwise document comparison requires the same MapReduce operations as an item-user comparison, similar results may be expected for benchmarking the calculations of a CB recommender.

For our experiment, we used Hadoop version 0.20.2, the latest stable release at the time of writing this paper, on a single Linux (Red Hat 4.1.2) machine. The machine has two quad-core processors running at 2.53GHz, 24GB memory and a solid state disk to save intermediate results. We utilized a subset of a static data set of Wikipedia articles, download from the Internet[1], as content items for the pairwise document comparison. The average file size of the articles in the experiment is 10kB. Since the articles are available in HTML format, the first mapper was adapted to filter out the stop words as well as the HTML-tags of the articles. In 20 successive iterations, the framework had to process a varying number of articles ranging from

[1] http://static.wikipedia.org/downloads/2008-06/en/

100 until 2000 with a step size of 100. After composing the term vectors with their corresponding term weights, $D*(D-1)/2$ similarities were calculated, where D is the number of input articles.

Detailed analysis of the required computation times learned that two jobs are responsible for more than 90% of the processing time, namely the first and the last job. The first job consists of reading the articles as well as filtering out the stop words and HTML tags. The last job calculates the cosine similarity for every vector pair. Based on this finding, we generated Figure 1, which shows the processing time spent on reading and filtering the articles, calculating the similarities, and the other jobs required to generate pairwise similarities. The total time spent on all jobs together is indicated in Figure 1 with "Total". The graph indicates that the time spent on reading the articles increases linearly, as the number of input files increases. In contrast, the time required for calculating the similarities shows, as expected, a quadratic increase for the successive iterations.

The calculation times of these jobs have a direct influence on the evolution of the total processing time. During the first iterations (i.e. iterations operating on less than 900 input files), the total processing time is dominated by the time required for reading and filtering the articles. As a result, the total processing time seems to increase linearly for the iterations with less than 900 input files. In contrast, if more than 900 input files have to be processed, the processing time needed for calculating the similarities exceeds the time spent on reading and filtering the articles. Given the quadratic increase of the processing time required for calculating the similarities during the successive iterations, the total processing time shows a quadratic increase too. This way, the total processing time evolves from a linear function to a quadratic function of the number of articles.

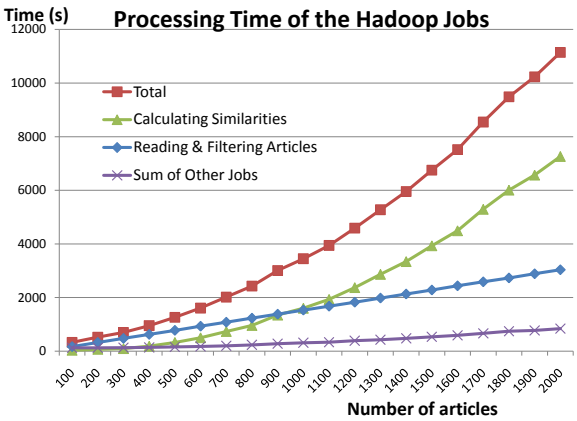

Figure 1: The processing time of the Hadoop jobs.

6 CONCLUSIONS

This paper explains in detail how Hadoop can be used to calculate content-based recommendations and pairwise item/user similarities in a scalable and reliable manner. Based on experiments with Wikipedia articles, performed on a single machine, we showed that for a limited number of input files, most processing time is spent on reading and filtering the articles. Consequently, the total processing time is a linear function of the number of processed items. Although if a larger number of input files have to be processed, the total processing time evolves to a quadratic function driven by the increasing processing time of the similarity calculations. In future research, we will benchmark the MapReduce operations on a cluster of multiple computing nodes. This way, we can investigate the true scalability potentials of the Hadoop framework.

REFERENCES

Brown, R. A. (2009). Hadoop at home: large-scale computing at a small college. In *SIGCSE '09: Proceedings of the 40th ACM technical symposium on Computer science education*, pages 106–110, New York, NY, USA. ACM.

Dean, J. and Ghemawat, S. (2008). Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113.

Elsayed, T., Lin, J., and Oard, D. W. (2008). Pairwise document similarity in large collections with mapreduce. In *HLT '08: Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies*, pages 265–268, Morristown, NJ, USA. Association for Computational Linguistics.

Ghemawat, S., Gobioff, H., and Leung, S.-T. (2003). The google file system. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 29–43, New York, NY, USA. ACM.

Lämmel, R. (2007). Google's mapreduce programming model — revisited. *Sci. Comput. Program.*, 68(3):208–237.

Mladenic, D. (1999). Text-learning and related intelligent agents: A survey. *IEEE Intelligent Systems*, 14(4):44–54.

Salton, G. and McGill, M. J. (1983). *Introduction to modern information retrieval*. McGraw-Hill computer science series. McGraw-Hill, New York, NY.