Supporting the efficient exploration of large-scale social networks for recommendation

A. Corbellini, C. Mateos, D. Godoy, A. Zunino, S. Schiaffino *

[1]ISISTAN Research Institute, CONICET-UNCPBA
Campus Universitario, Tandil (B7001BBO), Buenos Aires, Argentina

{acorbellini}@isistan.unicen.edu.ar

Abstract. *Most recommendation algorithms in the context of large-scale social networks struggle with the need of an efficient exploration of the underlying user graph. Current solutions in the form of graph-specific databases or frameworks for graph algorithms do not scale well for processing complex navigational patterns. In this paper we present an approach for supporting social recommendation algorithms that operate with large graphs in a computer cluster based on "policies", rules that allow users to throttle the amount of parallelism and control task location.*

1. Introduction

Social Recommender Systems (SRSs) aim to alleviate information overload over social media users by presenting the most attractive and relevant content, often using personalization techniques [5]. In the context of social networks, recommendation algorithms require the exploration of the underlying user graph in order to find information that is later summarized, ranked and presented to users in the form of suggestions (e.g., suitable friends to contact with, interesting posts to read, etc.).

For large-scale social networks such as Twitter or Facebook, the analysis of social relationships poses some challenges. In response, some developments in the form of graph-specific databases or frameworks for processing graph algorithms have arisen. However, these graph-oriented supports do not by themselves scale or perform efficiently in the presence of graph algorithms with complex navigation patterns, such as those for link prediction [9, 2].

The programmer is then responsible for adding logic to their algorithms in order to increase performance according to certain *policies*, e.g., taking advantage of data layout. In this paper we describe an approach for boosting the performance of these kind of large-scale graph algorithms via two policies, namely Balanced policy, which focuses on balancing computational load across nodes in the network infrastructure, and Location Aware policy, which exploits data locality and focuses on reducing network consumption.

Experimental evaluation of the proposed approach for supporting social recommendation was performed using a followee recommendation algorithm for Twitter that suggests other people a user might be interested in following [2]. In a first stage this algorithm explores the followee/follower network near the target user (i.e., the user receiving the suggestions) to select a set of candidate or potential users to recommend. We

*This work has been partially funded by ANPCyT, through project PICT-2011-0366, and CONICET, under grant PIP No. 114-200901-00381.

addressed the problem of supporting such search of candidates, independently of the later process of generating the actual recommendations.

The rest of this paper is organized as follows. Section 2 overviews the followee recommendation algorithm we used to test our approach. Section 3 describes the proposed approach. Section 4 summarizes the experiments carried out with the complete Twitter graph as of July 2009 [8]. Section 5 discusses related works and, finally, conclusions are stated in Section 6.

2. Followee Recommendation Algorithm

The algorithm used in this paper for evaluating the supporting infrastructure for exploring large social graphs is a followee recommender algorithm for Twitter users [2]. This algorithm is based on the characterization of Twitter users made in several studies such as [6], according to which users are mainly divided in two categories: information sources and information seekers. Users behaving as information sources tend to collect a large amount of followers as they are actually posting useful information or news, whereas information seekers follow several users to get information but rarely post a tweet themselves.

The hypothesis behind the algorithm is that the target user is an information seeker that has already identified some interesting users acting as information sources (i.e., his/her followees). Other people who also follow some of the users in this group share some interests with the target user and might have discovered other relevant information sources in the same topics (i.e., their followees). This last group are the candidates the algorithm needs to evaluate in the second phase.

The search of candidate users is performed according to the following steps:

Step 1. Starting with the target user u_T (an information seeker), obtain the list of users he/she follows (information sources). Let us call this list $S = \{s_1, s_2, \ldots, s_n\}$.

Step 2. For each element in S get its followers (information seekers), let us call the union of all these lists L, i.e., $L = \bigcup_{\forall s \in followees(u_T)} followers\,(s)$.

Step 3. For each element in L obtain its followees (information sources). Let us call the union of all these lists T, i.e., $T = \bigcup_{\forall l \in L} followees\,(l)$.

Step 4. Exclude from T those users that the target user is already following, resulting in a list of candidates R.

3. Middleware Support

In this section we present insights of the underlying structures used to isolate the algorithm from the networking details, such as message passing and code mobility. These structures effectively work as a thin middleware layer.

3.1. Ad-Hoc Distributed Key-Value Database

We used a distributed key-value database to store the adjacency list of each user in the Twitter graph. Although there are many open source databases that provide a key-value API [3, 11], we decided to create an ad-hoc distributed database for test purposes. The database client component exposes a simple *get/put* API, allowing an application to make queries to the database servers. The server component stores keys along with their corresponding values and responds to queries made by clients. To communicate clients with

servers we use JGroups[1], a network communication toolkit implemented in Java which follows P2P principles.

To find the corresponding server for a given key, every client applies a hashing function based on the number of available servers. The result of the hashing function is used as an index to access the server list and obtain a server. Then, the key-value pair is sent to the server to be stored in.

One of the requirements for the database is to persist the data after a shutdown. To achieve this, we implemented a storage layer on every server instance. The simplest way to provide storage support is by using an embedded data store. Among the existing data store alternatives, we chose LevelDB[2], a fast and lightweight embedded key-value store, to persist the data on every node. The main reason for choosing LevelDB is its sequential read performance. As the recommendation algorithm under study accesses keys sequentially most of the time, this type of storage is very convenient.

3.2. Parallel Processing Support

The original recommendation algorithm [2] was adapted to execute on top of GridGain, a Java-based parallel processing middleware. To achieve this, we identified parallelization opportunities in the algorithm and used them to create its GridGain version.

The implemented parallel algorithm consists of three types of tasks: T_S, T_L and T_T. Each task type corresponds to a different step in the algorithm:

- Tasks of type T_S: this type of task obtains the followees of the target user (step 1).
- Tasks of type T_L: this type of task obtains the list of followers of the users obtained by T_S (step 2).
- Tasks of type T_T: this type of task receives a list of followers from T_S and queries the followees of each users. Using this information, tasks build the list of recommended users along with the frequency of appearances of each user in the final stage (steps 3 and 4).

The algorithm works as follows. At startup time, it creates a task T_S^1 of type T_S using the target user as an argument. T_S^1 uses the given user to create a job J_S^1 of type J_S. This job obtains the list of followees of the user, and passes on the list to a new task T_L^1 of type T_L. T_L^1 divides the list of followees and creates jobs of type J_L. Each job J_L^i, where$i = 1..N_L$, obtains the list of followers of each input user and creates a new task T_T^i. In turn, each task of type T_T, receive a list of followers, divides this list and distributes a number of jobs of type J_T. These jobs collect the followees of each input user, count the number of appearances into a table, and return it to their parent task. Tasks merge the tables received by their children jobs into a final result. Finally, T_S^1 returns a table of users ranked by their number of appearances.

There are a number of alternatives to perform the division and mapping of jobs to computational nodes. In this work, we propose two:

- Balanced policy: this policy divides the amount of work by the number of available nodes, creating one job per node. Then it maps each job to a balanced node, i.e., a node that is relatively less occupied than the other nodes. To obtain a balanced node we use Grid Gain load balancing functionality.

[1] JGroups Web Page, http://www.jgroups.org/

[2] LevelDB Web Page on Google Code, https://code.google.com/p/leveldb/

- Location Aware policy: This policy takes advantage of the locality of the keys. It creates jobs by dividing the input into different lists of keys grouped by their physical location.

The main difference between policies is that Balanced policy focuses on a fair usage of computational nodes, whereas Location Aware policy focuses on a efficient usage of network resources.

4. Experiments

The experiments were carried out using a Twitter dataset[3] consisting of approximately 1400 million relationships between 40 million users [8]. This dataset only contains topological information about the social network, i.e., it only shows binary relationships between user IDs.

For selecting a test user group, we first filtered the list of users using the information source ratio [1], denoting to what extent the user can be considered an information source, defined as follows:

$$IS = \frac{followers(u)}{followers(u) + followees(u)}$$

We required that this ratio was lesser or equal than 0.5, which means that the number of followees had to be equal or larger than its number of followers. This restriction arises as a natural way of selecting which users would have any interest in receiving recommendations. After the initial filtering, we kept the top-5 users ordered by number of followees.

For each user, we ran the algorithm five times using the Balanced policy and five times using a Location Aware policy. For every run we collected the bytes sent over the network and the total recommendation time. Using this information we calculated the average number of bytes sent over the network and the average recommendation time.

Figure 1 summarizes the results obtained for a recommendation for the top-5 users (IDs) ordered by number of followees: 14389132, 14669398, 15991049, 16559157 y 17850012. As it can be observed from the table, the Location Aware policy gives a substantial improvement over the Balanced policy. With respect to the recommendation time, i.e., the time it takes to compute the R for a user, the Location Aware policy outperforms the Balanced approach by approximately 10 minutes. With respect to network consumption, the Location Aware policy generates approximately 6 times less traffic than the Balanced approach. graphically shows this comparison.

5. Related Work

Several alternatives can be found in the literature addressing the problem of storing and mining large-scale social data. For instance, Pregel [10] and Trinity [13] are closed-source graph processing frameworks created by Google and Microsoft, respectively. There are also a number of open-source projects for large-scale graph processing. As an example,

[3] http://an.kaist.ac.kr/traces/WWW2010.html

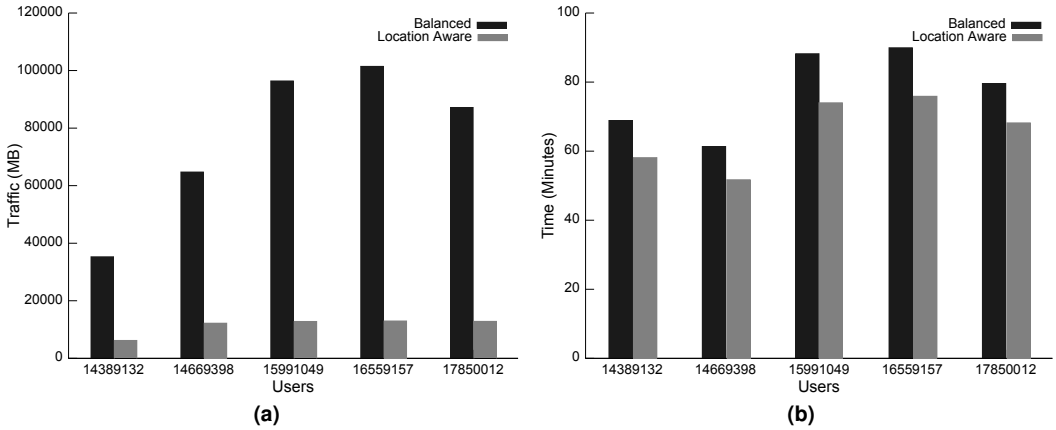

Figure 1. Comparison between Location Aware and Balanced policies w.r.t. Network Traffic and Recommendation Time.

HipG [7] is a graph processing framework that allows to model hierarchical parallel algorithms but focuses on the programming model for processing graphs rather than the way graph data is stored.

Previous studies have taken a similar approach to distribute a friendship recommendation algorithm. In [14], the authors use MapReduce to run topic analysis on microblogs obtained from a large Twitter dataset. The WTF algorithm [4] is used by Twitter to provide user recommendations. To achieve that, the algorithm maintains a "circle of trust", which contains the most relevant users in each user's network, and then obtains its followees.

6. Conclusions

In this paper, we presented an approach for supporting the efficient exploration of large-scale social networks required for social recommendation algorithms exhibiting complex graph navigation patterns. For testing this approach we used a MapReduce version of an existing recommendation algorithm for the Twitter social network and proposed two possible distribution *policies*: Location Aware Policy and a Balanced Policy.

In our experiments, the Location Aware Policy provides an improvement of approximately 15% on the time for recommendation over the Balanced Policy and 6 times less network usage. However, the Balanced Policy may be well suited for clusters were jobs of different applications must share the same set of nodes.

Future work includes a) the improvement of the underlying graph storage, b) the implementation of new policies and c) the testing of this approach with other graph-based recommendation algorithms. With respect to a), instead of a key-value NoSQL database, we will consider NoSQL databases designed for storing large graphs [12]. With regard to b), we are enhancing the Location Aware Policy with heuristics for selectively and smartly exploiting locality (e.g., depending on the size of the keys involved). Finally, with respect to c), we are currently working with link prediction algorithms such as SALSA [9].

References

[1] M. Armentano, D. Godoy, and A. Amandi. Towards a followee recommender system for information seeking users in Twitter. In *Proceedings of the International Workshop*

on Semantic Adaptive Social Web (SASWeb'11), Girona, Spain, 2011.

[2] M. Armentano, D. Godoy, and A. Amandi. Topology-based recommendation of users in micro-blogging communities. *Journal of Computer Science and Technology. Special Issue on Data Mining on Social Networks and Social Web*, 27(3):624–634, 2012.

[3] R. Cattell. Scalable SQL and NoSQL data stores. *ACM SIGMOD Record*, 39(4):12–27, 2011.

[4] P. Gupta, A. Goel, J. Lin, A. Sharma, D. Wang, and R. Zadeh. WTF: The who to follow service at Twitter. In *Proceedings of the 22th International World Wide Web Conference (WWW 2013)*, Rio de Janeiro, Brazil, 2013.

[5] I. Guy and D. Carmel. Social recommender systems. In *Proceedings of the 20th International Conference Companion on World Wide Web (WWW '11)*, pages 283–284, Hyderabad, India, 2011.

[6] A. Java, X. Song, T. Finin, and B. Tseng. Why we Twitter: Understanding microblogging usage and communities. In *Proceedings of the 9th WebKDD and 1st SNA-KDD 2007 Workshop on Web Mining and Social Network Analysis*, pages 56–65, San Jose, CA, USA, 2007.

[7] E. Krepska, T. Kielmann, W. Fokkink, and H. Bal. HipG: Parallel processing of large-scale graphs. *ACM SIGOPS Operating Systems Review*, 45(2):3–13, 2011.

[8] H. Kwak, C. Lee, H. Park, and S. Moon. What is Twitter, a social network or a news media? In *Proceedings of the 19th International Conference on World Wide Web (WWW'10)*, pages 591–600, Raleigh, NC, USA, 2010.

[9] R. Lempel and S. Moran. SALSA: The stochastic approach for link-structure analysis. *ACM Transactions on Information Systems (TOIS)*, 19(2):131–160, 2001.

[10] G. Malewicz, M. H. Austern, A. J. C. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: A system for large-scale graph processing. In *Proceedings of the 2010 International Conference on Management of Data (SIGMOD '10)*, pages 135–146, Indianapolis, IN, USA, 2010.

[11] R. P.Padhy, M. R. Patra, and S.C. Satapathy. RDBMS to NoSQL: Reviewing some next-generation non-relational databases. *International Journal of Advanced Engineering Science and Technologies*, 11(1):15–30, 2011.

[12] S. Sakr, A. Liu, D.M. Batista, and M. Alomari. A survey of large scale data management approaches in cloud environments. *Communications Surveys Tutorials, IEEE*, 13(3):311–336, 2011.

[13] B. Shao, H. Wang, and Y. Li. The Trinity Graph Engine. Technical Report MSR-TR-2012-30, Microsoft Research, March 2012.

[14] C. Zhang and J. Sun. Large scale microblog mining using distributed MB-LDA. In *Proceedings of the 21st International Conference Companion on World Wide Web (WWW '12 Companion)*, pages 1035–1042, Lyon, France, 2012.