A Parallel Algorithm for Tracking Dynamic Communities based on Apache Flink

Georgios Kechagias
School of Electrical and Computer Engineering
Technical University of Crete
Chania, Greece
gkechagias@isc.tuc.gr

Grigorios Tzortzis
George Paliouras
Institute of Informatics and Telecommunications
NCSR "Demokritos"
Athens, Greece
{gtzortzi,paliourg}@iit.demokritos.gr

Dimitrios Vogiatzis
[1]The American College of Greece, Deree
[2]Institute of Informatics and Telecommunications
NCSR "Demokritos"
Athens, Greece
dimitrv@iit.demokritos.gr

ABSTRACT

Real world social networks are highly dynamic environments consisting of numerous users and communities, rendering the tracking of their evolution a challenging problem. In this work, we propose a parallel algorithm for tracking dynamic communities between consecutive timeframes of the social network, where communities are represented as undirected graphs. Our method compares the communities based on the widely adopted *Jaccard similarity* measure and is implemented on top of Apache Flink, a novel framework for parallel and distributed data processing. We evaluate the benefits, in terms of execution time, that parallel processing brings to community tracking on datasets carrying different quantitative characteristics, derived from two popular social media platforms; Twitter and Mathematics Stack Exchange Q&A. Experiments show that our parallel method has the ability to calculate the similarity of communities within seconds, even for large social networks, consisting of more than 600 communities per timeframe.

CCS CONCEPTS

• **Human-centered computing → Social network analysis**; • **Computing methodologies → Parallel algorithms**;

KEYWORDS

Social Network Analysis, Community Tracking, Parallel Processing, Apache Flink

1 INTRODUCTION

In real world social networks, users who share common activities or interests (e.g. work, hobbies) closely interact (e.g. via likes, re-tweets) and form user communities. The membership of communities tends to change gradually as a result of users changing interests. As social interactions in these networks evolve, it is of great importance to develop analysis tools that effectively capture this evolution. In social network analysis, social networks are often represented as graphs, the social actors, or users, constitute the vertices of these graphs, and the edges indicate the social relationships or interactions between the network actors.

There exist two main directions in analyzing social networks represented as graphs. One is to handle them as static graphs which do not change as time passes. However, real life social networks, are dynamic and tend to evolve over time since their users dynamically join or leave communities. Hence considering the dynamic nature of communities in their analysis, in order to reveal their structure and evolution, is of great importance [1, 2, 5, 9, 10]. Studying the evolution of a community, which is the main focus of the dynamic community tracking literature [2, 4–8], consists of locating the instances (i.e. counterparts) of the community in the different timeframes, where each timeframe corresponds to a distinct time interval of a timestamped social network. Often an event is also assigned between two instances of the same community to designate the type of evolution (e.g. *community continuation*, *dissolution*).

Despite the recent bloom of methods tackling the problem of dynamic community tracking, one major issue of such approaches is the high number of comparisons required to calculate evolutionary events. In the typical case, community tracking algorithms have to compare every community of a given timeframe, with every community contained in the next one in order to uncover their counterparts and categorize their temporal evolution [4]. Apparently, this procedure does not scale well to large social networks.

In this work, we propose a simple and efficient parallel community tracking method, which compares communities of consecutive timeframes based on the *Jaccard similarity* measure. We implemented our method using Apache Flink[1], a framework for parallel and distributed data processing, which allows community comparisons to be executed across multiple CPUs. Subsequently, we evaluated our method using real world datasets derived from popular social media platforms such as Twitter[2] and Mathematics Stack Exchange Q&A[3]. Performing experiments using different Apache Flink configurations and datasets carrying different quantitative characteristics, enables the better assessment of the scalability of our method and of the behavior of Apache Flink under different loads.

SETN '18, July 9–15, 2018, Rio Patras, Greece
© 2018 Copyright held by the owner/author(s). Publication rights licensed to Association for Computing Machinery.
ACM ISBN 978-1-4503-6433-1/18/07…$15.00
https://doi.org/10.1145/3200947.3201039

[1]https://flink.apache.org/
[2]https://twitter.com/
[3]https://math.stackexchange.com/

2 PROBLEM DEFINITION

Before describing our framework, it is important to introduce the basic notation used in this work. A static undirected graph is denoted by $G = (V, E)$ where V is a set of vertices and $E \subseteq V \times V$ is a set of edges. A dynamic social network, is a collection of undirected graphs corresponding to the timeframes, that capture the time segmentation of the social network. This dynamic social network is defined as a set $TF = \{TF_0, TF_1, \ldots, TF_{T-1}\}$ of timeframes with T being the number of timeframes available. Each timeframe TF_t, contains a set of communities (i.e. subgraphs of the timeframe's graph) $\{C_t^0, C_t^1, \ldots, C_t^{K_t-1}\}$ with K_t denoting the number of communities in that timeframe. Each community C_t^k contains a set of vertices $V_{t,k}$ corresponding to the users of the community and a set of edges $E_{t,k}$ representing the interactions among its users.

The main problem addressed in this work, is the speed up of community tracking by parallelizing the execution of the comparisons required to match communities between sequential timeframes, using Apache Flink as the underlying parallelization framework. Each community C_t^k of a given timeframe TF_t must be compared to all communities C_{t+1}^j of the following timeframe. To compare communities, a community similarity measure is needed. As similarity measure we choose the *Jaccard similarity*, as it is very easy to calculate and has already been exploited in tracking communities [6, 7]. Given a pair of communities at consecutive timeframes C_t^k, C_{t+1}^j, their *Jaccard similarity* is defined as: $Jaccard(C_t^k, C_{t+1}^j) = \frac{V_{t,k} \cap V_{t+1,j}}{V_{t,k} \cup V_{t+1,j}}$.

3 APACHE FLINK

Apache Flink is an open source framework for distributed data streaming and artificial intelligence applications. It offers data structures and functions that help users in parallelizing their algorithms which then Flink is responsible to execute. Thus the details of parallel programming are abstracted away. Flink consists of three distributed components: (1) the *job client*, (2) the *job manager* and (3) the *task manager*. All these components have to communicate in order to parallelize and execute the submitted program. The *job client* takes a Flink job (i.e. an algorithm) as input, creates its *job graph* which is a representation of the algorithm's workflow, and submits it to the *job manager*. The *job manager* is responsible to determine the resource allocation and task scheduling of the program. When a *task manager* becomes available and has successfully registered with the *job manager*, the latter starts to distribute the individual tasks of a job to the *task manager*. The *task manager* is a JVM process which executes tasks using one or more threads. The *task manager* may have one or more available *task slots*, whose number *determines the degree of Flink's parallelism* of task execution. Tasks (e.g. *Map, Reduce, Cross*) are possible to be split into several subtasks which are distributed to the available task slots for execution, and each such subtask processes an appropriate subset of the task's input data. It is Flink's responsibility to decide the size of the input of each subtask. The number of parallel subtasks of a Flink task, is defined by the programmer by setting the number of task slots (i.e. the parallelism degree). At this point, it is important to mention that Flink by default utilizes all available CPU cores (even in the case of a single task slot) and that a task slot is not bound to a particular CPU core.

4 PARALLEL DYNAMIC COMMUNITY TRACKING

To track communities across consecutive timeframes, it is necessary to compare the communities by estimating their pairwise similarities, in order to locate their counterparts. The problem of calculating the similarities between two consecutive timeframes consisting of K and K' communities, respectively, has complexity $O(K \times K')$.

Due to the high number of comparisons required to perform community tracking, especially when using large datasets with several communities and timeframes, it is imperative to parallelize this procedure. In order to speed up community tracking, we use Apache Flink, a framework for parallel and distributed data processing. Note that in this work we solely focus on parallelizing the basic step involved in community tracking, i.e. the pairwise comparisons of communities. We do not study the steps involved in deciding which communities match and assigning evolutionary events between them. The matching and the assignment of events can be performed as a post-processing step using as input the community similarities. Hence, different tracking approaches can benefit from our parallelization.

Algorithm 1 Parallel Algorithm for Community Tracking

```
Input: $TF$ : the set of timeframes
1: Obtain Flink's execution environment
2: $TF_{vert} \leftarrow$Gather $V_{t,k}, \forall C_t^k \in TF_t, \forall t \in \{0, 1 \ldots T-1\}$
3: $TF_{ids} = \{0, 1 \ldots T-1\}$
4: $TF_{filt} = \{\}$
5: for $id \in TF_{ids}$ do
6:     $FilterResult \leftarrow Filter(TF_{vert}, id)$
7:     $TF_{filt} \leftarrow TF_{filt}.append(FilterResult)$
8: end for
9: $JaccardSim = \{\}$
10: for $i = 0$ to $SizeOf(TF_{filt}) - 2$ do
11:     $CrossResult \leftarrow Cross(TF_{filt,i}, TF_{filt,i+1}, Jaccard)$
12:     $JaccardSim \leftarrow JaccardSim.append(CrossResult)$
13: end for
Output: $JaccardSim$
```

An overview of our parallel method is described in Algorithm 1. After initializing Flink's execution environment (Step 1), we gather the vertices of each community into an array, and store these arrays into a new set (TF_{vert}) alongside with the their specific timeframe and community identifiers[4] (Step 2). This was done using the *GroupReduce* transformation of Apache Flink. In our method, for all timeframes, we grouped the edges of each community using the community id, and then we reduced the vertices of these communities to an array. Next (Steps 5-8), we filter each timeframe in the reduced dataset using its id. By filtering, an essential timeframe isolation is achieved since Flink's *DataSet*, an internal data structure which is used to store intermediate as well as final results, does not support indexes like those used in traditional programming languages. After that (Steps 10-13), by using Flink's *Cross* transformation, we automatically build all the pairwise combinations

[4]The timeframe identifiers are indexes from the set $\{0, 1, \ldots, T-1\}$. The community identifiers for timeframe TF_t are indexes from the set $\{0, 1, \ldots, K_{t-1}\}$.

Table 1: Number of communities and community vertices over the timeframes of the tested datasets.

	Communities			Vertices		
Dataset	min	max	mean	min	max	mean
Crimea	32	120	36	15	159	40
WorldCup	175	327	253	132	250	173
MathExchange	479	940	729	45	58	54

of the communities of two consecutive timeframes and apply on each pair the user-defined *Jaccard* similarity function. The Cross transformation ultimately builds the Cartesian product between the communities of two timeframes, which has the same complexity as a serial algorithm. The difference is that all these comparisons can now be executed over multiple CPU cores in a parallel fashion through Flink. Finally, when all *Cross* transformations are completed, Flink collects and returns the results (Step 14).

The main advantages of our method are that it can exploit all available CPUs without any effort due to Flink, an alternative similarity measure can be easily incorporated by simply replacing the similarity function and that the community matching and evolutionary events can be calculated at a post-processing step using the output of our algorithm.

5 DATASETS

To experimentally evaluate our community tracking method, we used three different datasets containing communities, derived from real world social networks. More specifically, we used two datasets from Twitter, one discussing the Crimea crisis on the 18th of March 2014, containing 208,841 tweets, and another discussing the 2014 FIFA World Cup during May 2014, containing 1,112,875 tweets. Furthermore we used a dataset containing communities from the Mathematics Stack Exchange Q&A website, which includes 376,030 posts spanning between 2009 and 2013. The two Twitter datasets were split in 20 timeframes, while the MathExchange dataset in 10 timeframes.

To detect communities on the Twitter datasets we employed the Louvain method [3] on the timeframe graphs. The vertices correspond to users who posted a tweet on a particular timeframe and an edge connects two users if either of the two mentioned the other in their tweets. In the MathExchange dataset each post is tagged with its subject areas (i.e. topics). To detect the communities in each timeframe, we take advantage of the topics and consider that users belong in the same community if they make posts about the same topic in a particular timeframe. Hence, each community is associated with a particular topic. Edges connect users who post an answer or comment to respond to the other's post.

The characteristics of the datasets used for our evaluation are summarized in Table 1. The Crimea dataset contains timeframes consisting of a relatively small number of communities, and these communities, contain on average a small number of vertices. The WorldCup dataset contains significantly more communities per timeframe compared to Crimea and these communities contain considerably more vertices. Finally, the MathExchange dataset consists of even more communities per timeframe than WorldCup, but these communities have on average less members than WorldCup.

6 EXPERIMENTAL EVALUATION

The evaluation of our community tracking method was performed using a machine with 12 cores at 2.5GHz and 30GB of RAM memory. Our parallel community tracking algorithm was implemented using the DataSet API offered by the Apache Flink framework and is available on Github[5]. The tracking algorithm described in Section 4 was executed for the Crimea, WorldCup and MathExchange datasets and the similarities among communities contained in consecutive timeframes were calculated. We performed experiments for different parallelism level configurations (i.e. number of task slots) of Apache Flink, ranging from parallelism 1 to parallelism 12 given that our machine had 12 CPUs, allowing us to locate the best configuration for each dataset. Remember (Section 3) that parallelism 1 (i.e. one task slot) does not imply serial execution.

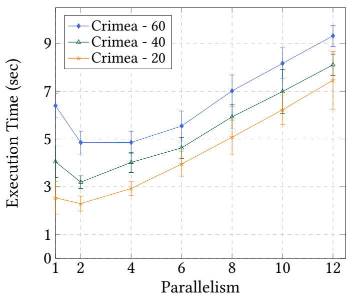
To better assess the scalability of our method in the context of a distributed framework like Apache Flink, we decided to conduct additional experiments by artificially enlarging each dataset. Specifically, we enlarged each dataset by repeating the existing timeframes two and three times. For the Crimea and WorldCup datasets we got two larger datasets containing 40 and 60 timeframes respectively, while for MathExchange two larger datasets containing 20 and 30 timeframes respectively. We repeated each experiment 25 times, and the mean execution time of these iterations is reported along with the standard deviation. The overall numerical results of our experiments are illustrated in Figure 1.

Using the Crimea dataset as input to our algorithm, as Figure 1a depicts, we can see that after an initial reduction of the execution time, it is clear that as parallelism increases, the performance is reduced. This behavior is observed to all versions of this dataset. There is a simple explanation behind this behavior. The Crimea dataset is a relatively small dataset, thus, by increasing parallelism (i.e. creating more subtasks and thus spawning more threads), the execution overhead also increases, becoming greater than the processing time needed for the actual data.

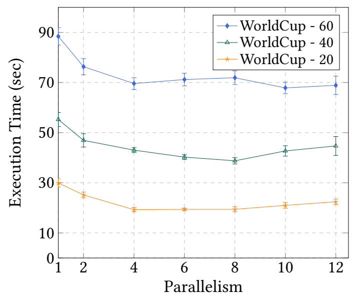
Experiments conducted using the WorldCup dataset, are presented in Figure 1b. For 20 timeframes, as parallelism increases, the execution of the algorithm decreases until parallelism 4. After that, the execution time remains steady, showing a slight increase for higher parallelism levels. By increasing the number of timeframes to 40, we can observe that our method's execution time now decreases until parallelism level 8. After this point, it is slightly increasing due to the overhead created by the additional parallelism. Finally, by increasing the input to 60 timeframes the execution time, drops until parallelism 4 and then remains steady with some slight fluctuations. We observe that as the dataset becomes larger (i.e. more timeframes) a higher parallelism level is preferable. An interesting observation throughout the above experiments, is the consistency of the results as expressed by the small standard deviation.

The results of our method on the MathExchange dataset are depicted in Figure 1c. Experiments conducted for 10 timeframes, show that until parallelism 6, the execution time slightly decreases. After that, the execution time remains relatively steady. However, for more timeframes, we observe a decrease in performance even for parallelism 4 and the standard deviation increases. Especially after parallelism 8, both for 20 and 30 timeframes experiments, a rapid

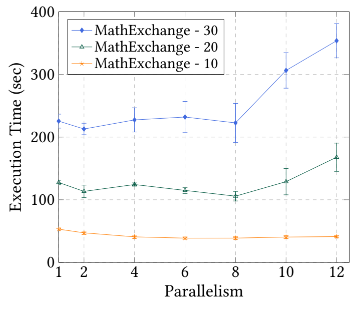
[5]https://github.com/gkech/A-Parallel-Algorithm-for-Tracking-Dynamic-Communities-Flink



(a) Crimea dataset results



(b) WorldCup dataset results



(c) MathExchange dataset results

Figure 1: Execution time of our community tracking approach for various Flink Parallelism levels on three social network datasets. The number next to the dataset name indicates the number of dataset timeframes.

Table 2: Execution time of GED algorithm.

Dataset	Exec. Time (sec)
Crimea 20	21.67
WorldCup 20	810.45
MathExchange 10	1670.69

drop in performance is evident, and the standard deviation further increases. We observed that experiments with 20 and 30 timeframes of the MathExchange dataset, utilized fully the machine's RAM memory, which created a severe bottleneck in the execution of the experiments and introduced delays that are related to memory management issues and not to our method.

Until now, we discussed only the execution time variations observed for different parallelism levels of Apache Flink. Below we compare our method to a popular serial community tracking algorithm, in order to get a rough estimate of the speedup we achieve. Table 2 illustrates the execution time needed to calculate evolutionary events of communities in consecutive timeframes using the GED method [4] implemented in Python[6], on the aforementioned datasets. We chose GED for this comparison as it follows a very similar approach to ours in order to track the communities. Even for parallelism 1, our method achieves execution time which is 88.3% faster than the sequential GED on the Crimea dataset. In fact, for the two bigger datasets of WorldCup and MathExchange, this improvement increases to 96.3% and 96.8% respectively.

Summarizing the reported results, our experiments indicate that the parallel algorithm proposed in this work is much faster than serial algorithms for tracking dynamic communities. Moreover, the best parallelism level of Apache Flink, depends on the underlying dataset, as depicted in Figure 1. It is clear that for larger datasets with more timeframes and communities per timeframe, higher parallelism is preferable. It is also clear from the experiments that a high parallelism level may have negative impact on performance if the dataset is not big enough, as in the case of the Crimea dataset.

7 CONCLUSIONS

In this paper, we presented a parallel algorithm for tracking dynamic communities based on Apache Flink. Our proposed method has the ability to calculate the similarity of communities contained in consecutive timeframe within seconds, using a widely adopted similarity measure. The evaluation of our method was conducted using three real world social networks, which enabled the comprehensive evaluation of the scalability of our method with very encouraging results. Our method is highly modular, as the similarity function used to compare communities, could be easily substituted. Furthermore, as we implemented this method using Apache Flink, it has the ability to scale to larger datasets to those tested here.

In the future we intend to extend our parallel algorithm, by incorporating the necessary steps to categorize the evolution of communities using the event labels proposed in the literature, such as *community continuation, growth, shrinkage* etc. Additionally, we plan to devise a software suite for dynamic community tracking which is going to offer streaming capabilities in order to track evolutionary events in real time on top of the fast batch data processing proposed in this work.

[6]https://github.com/iit-Demokritos/community-Tracking-GED

REFERENCES

[1] Charu C. Aggarwal and Karthik Subbian. 2014. Evolutionary Network Analysis: A Survey. *ACM Comput. Surv.* 47, 1 (2014), 10:1–10:36.
[2] Sitaram Asur, Srinivasan Parthasarathy, and Duygu Ucar. 2007. An event-based framework for characterizing the evolutionary behavior of interaction graphs. In *ACM SIGKDD.* 913–921.
[3] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. 2008. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment* 2008, 10 (2008), P10008.
[4] Piotr Bródka, Stanislaw Saganowski, and Przemyslaw Kazienko. 2013. GED: the method for group evolution discovery in social networks. *Social Netw. Analys. Mining* 3, 1 (2013), 1–14.
[5] Georgios Diakidis, Despoina Karna, Dimitris Fasarakis-Hilliard, Dimitrios Vogiatzis, and George Paliouras. 2015. Predicting the Evolution of Communities in Social Networks. In *WIMS.* 1:1–1:6.
[6] Mark K. Goldberg, Malik Magdon-Ismail, Srinivas Nambirajan, and James Thompson. 2011. Tracking and Predicting Evolution of Social Communities. In *PASSAT/SocialCom.* 780–783.
[7] Derek Greene, Dónal Doyle, and Padraig Cunningham. 2010. Tracking the Evolution of Communities in Dynamic Social Networks. In *ASONAM.* 176–183.
[8] Gergely Palla, Albert-László Barabási, and Tamás Vicsek. 2007. Quantifying social group evolution. *Nature* 446, 7136 (2007), 664–667.
[9] Maria Evangelia G. Pavlopoulou, Grigorios Tzortzis, Dimitrios Vogiatzis, and George Paliouras. 2017. Predicting the evolution of communities in social networks using structural and temporal features. In *SMAP.* 40–45.
[10] Etienne Gael Tajeuna, Mohamed Bouguessa, and Shengrui Wang. 2015. Tracking the evolution of community structures in time-evolving social networks. In *DSAA.* 1–10.