A Parallel Library for Social Media Analytics

Conference Paper · July 2017

DOI: 10.1109/HPCS.2017.105

4 authors:

L. Belcastro
Università della Calabria
18 PUBLICATIONS **56** CITATIONS
SEE PROFILE

Fabrizio Marozzo
Università della Calabria
60 PUBLICATIONS **483** CITATIONS
SEE PROFILE

Domenico Talia
Università della Calabria
407 PUBLICATIONS **5,113** CITATIONS
SEE PROFILE

Paolo Trunfio
Università della Calabria
139 PUBLICATIONS **1,941** CITATIONS
SEE PROFILE

A Parallel Library for Social Media Analytics

Loris Belcastro, Fabrizio Marozzo, Domenico Talia, Paolo Trunfio
DIMES Department, University of Calabria,
Rende, Italy
Email: [lbelcastro, fmarozzo, talia, trunfio]@dimes.unical.it

***Abstract*—Social media analysis is a fast growing research area aimed at extracting useful information from huge amounts of data generated by social media users. This work presents a Java library, called ParSoDA (Parallel Social Data Analytics), which can be used for developing parallel data analysis applications based on the extraction of useful knowledge from large dataset gathered from social networks. The library aims at reducing the programming skills necessary to implement scalable social data analysis applications. To reach this goal, ParSoDA defines a general structure for a social data analysis application that includes a number of configurable steps, and provides a predefined (but extensible) set of functions that can be used for each step. The paper describes the ParSoDA library and presents two case studies to assess its usability and scalability.**

***Index Terms*—Social Data analysis, Scalability, MapReduce, Cloud computing, Parallel library, Big Data**

I. Introduction

The large volumes of data generated in social networks, such as Facebook, Twitter, Foursquare and Flickr, can be exploited to extract valuable information about human dynamics and behaviors. Social media analysis is a fast growing research area aimed at extracting useful information from this big amount of data [1]. It is used for the analysis of collective sentiments [2], for understanding the behavior of groups of people [3][4] or the dynamics of public opinion [5]. To cope with the size and complexity of social media data, the use of parallel and distributed data analysis techniques is essential. Despite the wide availability of powerful parallel frameworks [6], first and foremost MapReduce [7], it is still hard for many users to use such frameworks, mainly due to the programming skills necessary to implement the desired data analysis methods on top of them.

To address this problem, we created ParSoDA (Parallel Social Data Analytics), a Java library that can be used for building parallel data analysis applications that extract useful knowledge from social media data. ParSoDA was designed with the primary goal of reducing the programming skills necessary to implement the desired data analysis application. To this end, ParSoDA includes functions that are widely used to process and analyze data gathered from social media for finding different types of information (e.g., user mobility, user sentiments, topics trends). The project pays particular attention to providing functions dealing with Big Data. For this reason, most algorithms are based on the MapReduce model and can be executed in parallel on distributed systems, such as the Cloud.

ParSoDA defines a general structure for a social data analysis application that includes a number of steps (data acquisition, filtering, mapping, partitioning, reduction, analysis, and visualization), and provides a predefined (but extensible) set of functions for each step. Thus, an application developed with ParSoDA is expressed by a concise code that specifies the functions invoked at each step. In this way, data analysts having limited programming skills, especially with regards to parallel programming, can efficiently design and execute data analysis applications dealing with big amounts of social media data. The library includes algorithms that are widely used on social media data for extracting different kinds of information. To deal with social media items gathered from different social networks, ParSoDA defines a metadata model that represents the different types of social media items (tweets, Flickr posts, etc.). The model can be easily extended to match most application requirements.

In this paper, we present the main features of ParSoDA and describe how it can be used to create data analytics applications dealing with Big Data collected from social networks. To assess the usability and scalability of ParSoDA, we present two case study applications that make use of the library to extract sequential patterns and frequent itemsets from social media data published in Flickr and Twitter. The first application aims at discovering sequential patterns from user movements, so as to find the common routes followed by users. The goal of the second application is to discover the frequent sets of places visited by users. The scalability was evaluated carrying out the data analysis applications on a Cloud platform. Using a Hadoop cluster with 2 head and 12 worker nodes, we obtained an almost linear speedup.

The remainder of the paper is organized as follows. Section II discusses related work. Section III describes the ParSoDA library. Section IV presents two cases of study and the results obtained. Finally, Section V concludes the paper.

II. Related work

Several research projects have been carried out to develop applications and algorithms for extracting useful information from data extracted out of social networks (e.g., Flickr, Foursquare, Twitter, Facebook). In most cases the amount of data to be analyzed is so big that high performance computers, such as many and multi-core systems, Clouds, and multi-clusters, paired with parallel and distributed algorithms, are used by data analysts to reduce response time to a reasonable value [8].

Several research activities focused not only on data analysis, but also on providing solutions for building social data applications, with the aim of helping scientists to develop the different steps that compose social data mining applications without the need to implement common operations from scratch.

Perisikan [9] is a framework designed for social network analysis over Facebook that is composed by several components for data acquisition through API, filtering, cleaning, parsing and stemming. The system has also a data analysis and pattern matching engine that performs data modeling and indexing, using machine learning, text and image processing algorithms. The framework has been tested with a limited amount of data, so no tests about the performance and usability in presence of huge amounts of data have been performed.

SOCLE [10] is a framework for expressing and optimizing data preparation in social applications. It is composed by a general-purpose three-layers architecture, an algebra, and a language to define operations for data preparation in social applications. As an example, SOCLE provides operators to remove all unnecessary information from data (data pruning), to add information by using external sources (data enrichment), to transform data values (data normalization). The authors examined the use of SOCLE for manipulating social data in two families of social applications, recommendation and analytics, but no studies have been performed to assess its scalability, and no details about framework requirements have been provided.

Cuesta et al. [11] proposed a framework for easing Twitter data extraction and analysis. In the proposed architecture the tweets, mined by the application through the Twitter APIs, are cleaned and then stored in a MongoDB database [12]. In addition to basic database operations (i.e. selection, projection, insertion, updating and deletion), the framework can be extended creating more complex aggregation MapReduce tasks in Python. By default, the framework provides researchers modules for executing sentiment analysis and generating reports.

SODATO (SOcial Data Analytics Tool) [13] is an on-line tool for helping researches on social data. It utilizes the APIs provided by social networks (i.e., currently, it supports only Facebook and Twitter) for collecting data; then, it provides a combination of web as well as console applications that run in batches for preprocessing and aggregating data for analysis. At the end of the analytics process, the results can be displayed using the integrated visualization module. SODATO provides methods for several kinds of analysis, such as sentiments analysis, keyword analysis, content performance analysis, social influencer analysis, etc.

Stieglitz and Dang-Xuan [14] proposed a framework that provides appropriate methods and techniques required for tracking, monitoring and analyzing content from social media in political context. It works with three type of social media: microblogging, social network, and weblogs. To prepare textual data for further analysis, the framework provides a set of preprocessing functions, such as stop words filtering, stemming, and lemmatization. Regarding data analytics, the framework provides a set of methods for sentiment/option, trend, and content analysis. As application cases, the framework can be used to execute analysis for detecting: the most emerging political topics that are relevant for own reputation, political relevant communities or leaders, prevalent sentiment or option related to a specific topic, etc.

You et al. [15] presented a framework, running on Clouds, for developing social data analysis applications for smarter cities, especially designed to support smart mobility. In particular, the framework is composed by five components (i.e., data collector, data preprocessor, data analyzer, data presenter, and data storage) that cover the whole data analysis lifecycle. The framework supports data collection from social networks (e.g., Twitter, Foursquare), by exploiting their public APIs, and from other Internet sources (e.g. website, blog, files). A component devoted to data preprocessing provides functions for data cleansing, filtering and normalization. Afterwards, the data analyzer component provides needed analysis methods (e.g. K-means, DBScan, and Self-organizing Map) to make some data analysis.

The main differences between ParSoDA and the systems described above (but the one by You et al. [15]), is that our system was specifically designed to build Cloud-based data analytics applications. To this end, it provides scalability mechanisms based on the MapReduce model, which are fundamental to provide satisfactory services as the amount of data to be managed grows. However, differently from [15], ParSoDA is available as an open-source library, which allows programmers developing their own applications with a high degree of flexibility.

III. The ParSoDA library

As mentioned before, ParSoDA (Parallel Social Data Analytics) is a Java library that includes algorithms that are widely used to process and analyze data gathered from social networks for extracting different kinds of information (e.g., user mobility, user sentiments, topic trends).

ParSoDA defines a general structure for a social data analysis application that is formed by following steps:

- *Data acquisition*: during this step, it is possible to run multiple crawlers in parallel; the collected social media items are stored on a distributed file system (HDFS [16]).
- *Data filtering*: this step filters the social media items according to a set of filtering functions.
- *Data mapping*: this step transforms the information contained in each social media item by applying a set of map functions.
- *Data partitioning*: during this step, data is partitioned into shards by a primary key and then sorted by a secondary key.
- *Data reduction*: this step aggregates all the data contained in a shard according to the provided reduce function.
- *Data analysis*: this step analyzes data using a given data analysis function to extract the knowledge of interest.

- *Data visualization*: at this final step, a visualization function is applied on the data analysis results to present them in the desired format.

For each of these steps ParSoDA provides a predefined set of functions. The users are free to extend these functions with their own. For example, for the data acquisition step, ParSoDA provides crawling functions for gathering data from some of the most popular social networks (Twitter and Flickr), while for the data filtering step, ParSoDA provides functions for filtering geotagged items based on their position, time of publication, and contained keywords.

A. Reference architecture and execution flow

Figure 1 presents a reference architecture that describes how user applications based on the ParSoDA library are executed on the popular Hadoop MapReduce framework [17], which allows implementing parallel and distributed applications with high level of scalability for several data mining tasks [18]. As shown in the figure, user applications can make use of ParSoDA and other MapReduce libraries (e.g., Mahout[1], Storm[2], Giraph[3]). Applications are executed on a Hadoop cluster, using YARN as resource manager and HDFS as distributed file system.

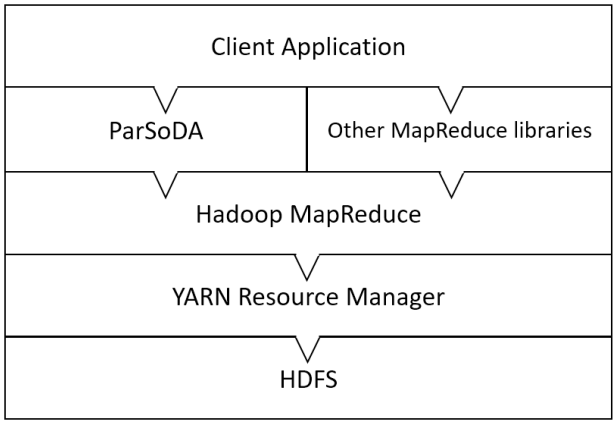


Fig. 1. Reference architecture

Figure 2 provides details on how applications are executed on a Hadoop Cluster. The cluster is formed by one or more master nodes, and multiple slave nodes. Once a user application is submitted to the cluster, its steps are executed according to their order (i.e., data acquisition, data filtering, etc.).

Some steps are inherently MapReduce-based, namely: data filtering, data mapping, data partitioning and data reduction. This means that all the functions used to perform these steps are executed within a MapReduce job that runs on a set of slave nodes. Specifically: the data filtering and data mapping steps are wrapped within Hadoop Map tasks; the data partitioning step corresponds to Hadoop Split and Sort tasks; the data reduction step is executed as a Hadoop Reduce task.

[1]http://mahout.apache.org/
[2]http://storm.apache.org/
[3]http://giraph.apache.org/

The remaining steps (data acquisition, data analysis, and data visualization) are not necessarily MapReduce-based. This means that the functions associated to these steps could be executed in parallel on multiple slave nodes, or alternatively they could be executed locally by the master node(s). The latter case does not imply that execution is sequential, because a master node could make use of some other parallel runtime (e.g., MPI).

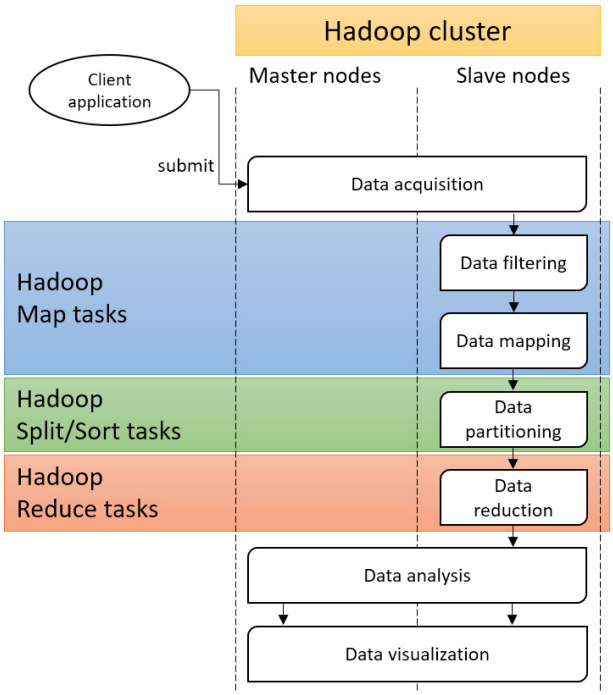


Fig. 2. Execution flow.

B. Metadata model for social media data

To deal with social media items gathered from different social networks, ParSoDA defines a metadata model for representing the different types of social media items (tweets, Flickr posts, etc.). According to this model, each social media item is represented by a metadata document composed of two parts: a *basic* section that includes fields common to all social networks (source, item id, date and time, location coordinates, user info); an *extra* section that contains fields specific to the source. As an example, Listing 1 shows a metadata element describing a tweet. The *source* field indicates that it is a social media item gathered from Twitter, and therefore the *extra* section contains fields specific to the tweets (whether it is a retweet or not, the retweet count, and so on).

```
{
 "BASIC":{
  "SOURCE":"Twitter",
  "ID":"111222333444555",
  "DATETIME":"2015-12-20T23:20:34.000",
  "LOCATION":{"LNG":-0.1262,"LAT":51.5011},
  "USER":{ "USERID":"12345", "USERNAME":"
    joedoe"}
```

```
 },
 "EXTRA":{
  "inReplyToScreenName":"billsmith",
  "inReplyToUserId":123456789,
  "hashtags":[ "#code", "#mapreduce"],
  "inReplyToStatusId":678712345678962848,
  "text":"@billsmith that sounds great!",
  "retweets":0,
  "isRetweet":false
 }
}
```

Listing 1. Metadata of a tweet serialized in JSON format.

ParSoDA defines an abstract class named *SocialItem* that defines the *basic* fields, and a set of classes (*TwitterSocialItem*, *FlickrSocialItem*, etc.) that extend *SocialItem* by defining the *extra* fields specific to different social networks. Each social media item is represented in memory by an instance of one such classes (e.g., a tweet will be an instance of *TwitterSocialItem*). When the metadata of a social media item must be saved to persistent storage or sent through the network, the object is serialized in JSON format, a widely-used text notation [19].

C. Structure of a ParSoDA application

As mentioned earlier, ParSoDA defines a general structure for a social data analysis application that includes a number of steps (data acquisition, filtering, mapping, partitioning, reduction, analysis, and visualization), and provides a predefined (but extensible) set of functions for each step. Thus, an application developed with ParSoDA is expressed by a concise code that specifies the functions invoked at each step. More specifically, a ParSoDA application can be developed by creating an instance of a class named *SocialDataApplication*, which defines a set of methods that allow the programmer specifying the functions to be used at each step.

Table I lists the main methods of the *SocialDataApplication* class. For each method, the table specifies the step it refers to, and a short description.

For the *Data acquisition* step, the *SocialDataApplication* class provides the *setCrawlers* method that can be used to specify which crawling functions will be used to collect data from social networks. The method receives two arrays of strings as parameters: *functions* and *params*. Array *functions* contains the fully-qualified names of the crawling classes that will be instantiated to perform data collection. The Java reflection mechanism is used to create instances of a class from its fully-qualified name. Array *params* contains the parameters that are necessary to configure the instances of the crawling classes specified in *functions*; specifically, *params[i]* contains the configuration string of *functions[i]*. In ParSoDA, a set of crawling classes are available. For example, a *FlickrCrawler* class can be instantiated to collect data from the Flickr social network. If *functions* specifies multiple crawling classes, they will be instantiated and run in parallel.

Data filtering can be configured with the *setFilters* method. It works similarly to the *setCrawlers* method. In fact, its first parameter can be used to specify the names of the classes that will be instantiated to perform data filtering, while the second one contains the parameters used to configure the instances of the filtering classes. ParSoDa implements a simple but effective mechanism for filtering social media items according to a set of conditions. Each filtering class implements a predicate function that verifies if a social media item meets or not a particular condition. In ParSoDA, a filtering class is defined by implementing the interface *Predicate*, which is included in Java since version 1.8. After executing the data filtering step, only social media items that match all conditions provided will be passed to the data mapping step.

For configuring the *Data mapping* step, the developer has to use the *setMapFunctions* method. Similarly to methods

TABLE I
MAIN METHODS OF THE *SocialDataApplication* CLASS.

Step	Function	Description
Data acquisition	*setCrawlers(String[] functions, String[] params)*	Specifies the crawling functions to be used for data acquisition. The *functions* array contains the fully-qualified name of the crawling classes; *params[i]* contains the configuration string of *functions[i]*.
Data filtering	*setFilters(String[] functions, String[] params)*	Specifies the functions and associated parameters to be used to perform data filtering.
Data mapping	*setMapFunctions(String[] functions, String[] params)*	Specifies the functions and associated parameters to be applied at the mapping step.
Data partitioning	*setPartitioningKeys(String groupKey, String sortKey)*	Specifies the keys used by the *secondary sort design* pattern, which partitions data into shards by a primary key (*groupKey*) and then sorts all data in a shard by a secondary key (*sortKey*).
Data reduction	*setReduceFunction(String function, String params)*	Specifies the function and associated parameters to be used at the reduction step.
Data analysis	*setAnalysisFunction(String function, String params)*	Specifies the function and associated parameters to be used to perform data analysis.
Data visualization	*setVisualizationFunction(String function, String params)*	Specifies the function and associated parameters to be used for data visualization.

described above, it receives two arrays of strings as parameters, which specify, respectively, the names of the classes that will be instantiated to perform data mapping and the parameters used to configure them. In ParSoDa a mapping class defines a function that transforms a social media item given in input. In such way, developers are able to transform social media items by applying a sequence of map functions. A map function can be defined by extending the abstract class *MapFunction*. Also in this case, the Java reflection mechanism is used to create instances of a class from its fully-qualified name.

Data partitioning can be configured with the *setPartitioningKeys* method, which receives two strings as parameters: *groupKey* and *sortKey*. The method partitions data in shards by *groupKey* and then sorts all data in a shard by *sortKey*. The keys used to configure this step must be present in the metadata model used to represent the social media items under processing. ParSoDA implements the *Secondary Sort* design pattern [20], which allows configuring a primary key (*groupKey*) for partitioning data into shards, and a secondary key (*sortKey*) for sorting all data in a shard. As an example, this mechanism can be used to partition data by user ids and then to sort it by timestamps, which is a very common task in sequential pattern mining.

Data reduction can be configured with the *setReduceFunction* method, which receives as parameters the name of the class that will be instantiated to perform data reduction and the parameters used to configure it. The reduce function aggregates all the data contained in a shard. As an example, to analyze movements of social media users, one might use a reduce function for aggregating all the data of a single user according to given criteria. In ParSoDA, a reduce function can be defined by creating a class that implements the interface *ReduceFunction*.

Data analysis is configured with the *setAnalysisFunction* method, which receives as parameters the name of the class that will be instantiated to perform the data analysis task and the associated parameters. A data analysis function can be defined by extending the abstract class *AnalysisFunction*, which requires the implementation of two abstract methods: *formatData*, for formatting the input data in the format required by the analysis function, and *analyzeData* that implements the data analysis algorithm.

Finally, *Data visualization* can be configured with the *setVisualizationFunction* method that, similarly to the previous methods, receives the name of the data visualization class and the parameters required to create an instance. To create a custom data visualization function, the programmer must define a class that implements the interface *VisualizationFunction*.

IV. Study cases

To evaluate the usability and scalability of ParSoDA, we used it to develop two social data analysis applications that extract sequential patterns and frequent itemsets. The first application aims at discovering sequential patterns from user movements, so as to find the common routes followed by users. The second one aims at discovering the frequent sets of places visited by users. The analysis was carried out by analyzing 325 GB of social media data published in Flickr and Twitter from November 2014 to July 2016 that refer to the center of Rome.

A. Application code

Listing 2 shows the code of the application for executing the sequential pattern mining. First, an instance of the *SocialDataApp* class must be created (*line 1*). Then a file containing the boundaries of the regions of interest (*RomeRoIs.kml*) is distributed to the processing nodes (*lines 2-3*). Afterwards, the different steps of the application are configured as described here:

1) *Data collection*. The names of two crawling classes (*FlickrCrawler* and *TwitterCrawler*) are defined in the *cFunctions* array (*line 4*). The parameters used to configure the instances of the two crawling classes are defined in the *cParames* array (*line 5*). The two arrays are then passed to the *setCrawlers* method (*line 6*).
2) *Data filtering*. Two filtering classes are specified: *IsGeotagged* and *IsInPlace* (*line 7*). The former filters data by keeping only geotagged items. The latter filters out data that are not in the center of Rome, which is defined by its geographical coordinates. The parameters of the two filtering functions are specified in the *fParams* array (*line 8*). The names of the filtering classes and associated parameters are then passed to the *setFilters* method (*line 9*).
3) *Data mapping*. The map class *FindPoI* (*line 10*), which does not require parameters to be instantiated (*line 11*), is specified. The mapping function defined in *FindPoI* assigns to each social media item the name of the place it refers to. To do this, it refers to the boundaries specified in the file defined at *line 2*. The name of the map class is then passed to the *setMapFunctions* method (*line 12*).
4) *Data partitioning*. The id of the user who posted a social media item is used as the *groupKey* (*line 13*), while the date and time when the social media item was posted is used as the *sortKey* (*line 14*). The two keys are then passed to the *setPartitioningKeys* method (*line 15*).
5) *Data reduction*. A reduce class, named *ReduceByTrajectories* (*line 16*), is specified to aggregate all the social media items posted by a single user, into a list of individual trajectories across places. The parameters of the reduce class are specified in the *rParams* string (*line 17*). In particular, it receives only a parameter t, which is the maximum time gap in hours that can be taken for consecutive places in the same trajectory. The output of the reduce function will be a list of trajectories $\{T_1, T_2, ..., T_n\}$, where $T_i = \{p_{i1} \rightarrow p_{i2} ... \rightarrow p_{in}\}$, and p_{ij} is the name of the *j*-th place visited by the user in the *i*-th trajectory. The name of the reduce class and its parameters are then passed to the *setReduceFunction* method (*line 18*).
6) *Data analysis*. A data analysis class, named *MGFSM*, is specified (*line 19*). The class implements MG-FSM [21],

a scalable frequent sequence mining algorithm built for MapReduce that takes as input a collection of sequences and mines frequent sequences. The parameters of data analysis class are specified in the *aParams* string (*line 20*). The name of the data analysis class and its parameters are then passed to the *setAnalysisFunction* method (*line 21*).

7) *Data visualization*. The *SortResults* class is specified to perform the data visualization function (*line 22*). A configuration string *vParams*, containing the parameters of the data visualization class, is specified at *line 23*. The class receives two parameters: the key used to sort results (the sequence support) and the sort direction (descending order). The name of the data visualization class and its parameters are then passed to the *setVisualizationFunction* method (*line 24*).

Finally, the execution of the application is obtained by invoking the *execute* method (*line 25*).

```
1  SocialDataApp app = new SocialDataApp("SPM - City
     of Rome");
2  String[] cFiles = {"RomeRoIs.kml"};
3  app.setDistributedCacheFiles(cacheFiles);
4  String[] cFunctions = {"FlickrCrawler","
     TwitterCrawler"};
5  String[] cParams = {"-lat 12.492 -lng 41.890 -
     radius 10 -startDate 2016-07-31 -endDate
     2014-11-01","-lat 12.492 -lng 41.890 -radius 10
     -startDate 2016-07-31 -endDate 2014-11-01"};
6  app.setCrawlers(cFunctions,cParams);
7  String[] fFunctions = {"IsGeotagged","IsInPlace"};
8  String[] fParams = {"true","-lat 12.492 -lng
     41.890 -radius 10"};
9  app.setFilters(fFunctions, fParams);
10 String[] mFunctions = {"FindPoI"};
11 String[] mParams = null;
12 app.setMapFunctions(mFunctions, mParams);
13 String groupKey = "USER.USERID";
14 String sortKey = "DATETIME";
15 app.setPartitioningKeys(groupKey,sortKey);
16 String rFunction = "ReduceByTrajectories";
17 String rParams = "-t 5";
18 app.setReduceFunction(rFunction,rParams);
19 String aFunction = "MGFSM";
20 String aParams = "-m d -g 3 -l 5 -s 50";
21 app.setAnalysisFunction(aFunction,aParams);
22 String vFunction = "SortBy";
23 String vParams = "-k support -d DESC";
24 app.setVisualizationFunction(vFunction,vParams);
25 app.execute();
```

Listing 2. An example of sequential pattern mining (SPM) application on Flickr and Twitter data from the City of Rome, written using the ParSoDA library.

The code for executing the frequent itemset analysis differs from that described above only for the used data analysis algorithm (*lines 19-21*). In particular, to extract frequent sets of places from social media data, a parallel implementation of the FP-Growth algorithm [22], called PFP [23], has been used.

B. Analysis results

Figure 3(a) shows 24 popular places in the center of Rome that have been considered to run the sequential pattern mining task and the frequent itemset discovery task, both implemented as ParSoDA applications. In the following, we discuss some of the most interesting results that have been obtained. Table II shows the top 5 places visited in Rome, with the corresponding support in the data. The Colosseum is the most visited place, followed by the St. Peter's Basilica.

TABLE II
TOP 5 PLACES VISITED IN ROME

Place	Support
Colosseum	21.7%
St Peter's Basilica	13.9%
Trastevere	8.7%
Pantheon	6.5%
Trevi Fountain	5.3%

Table III shows the most frequent itemsets of length 3 that have been discovered by the PFP algorithm. Set {*Pantheon, St. Peter's Basilica, Colosseum*} is the most frequent set of places visited by social users in Rome, with a support of 5.3%. Combining the information contained in Tables II and III, an interesting result is that *Trastevere*, a popular district of Rome, is the third most visited place, but it is not present in any frequent itemset. This could happen because Trastevere is visited by people during the evening, for having a dinner in one of its many restaurants or pubs, but it is not part of common tourist routes during the daylight.

TABLE III
TOP 5 FREQUENT SETS OF PLACES VISITED IN ROME

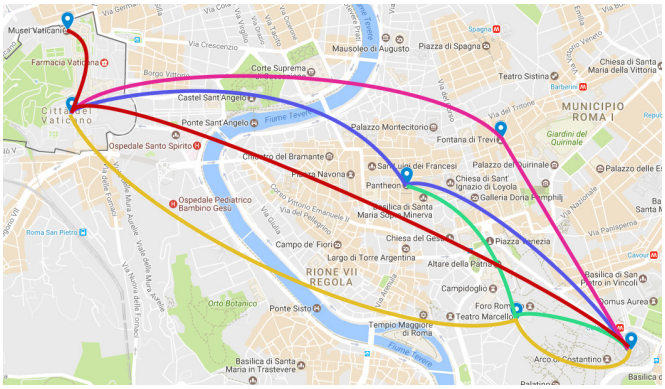
Set of places	Support
Pantheon, St. Peter's Basilica, Colosseum	5.3%
Trevi Fountain, St. Peter's Basilica, Colosseum	4.5%
Roman Forum, St. Peter's Basilica, Colosseum	4.4%
Vatican Museums, St. Peter's Basilica, Colosseum	4.4%
Trevi Fountain, Pantheon, Colosseum	4.0%

The sequential pattern analysis has been carried out for discovering the most frequent routes in Rome. In this experiment, it has been set a maximum time duration (gap) to move from a place to another of 5 hours. This means that if the time distance between two contiguous places in sequence is greater than 5 hours, they will belong to different sequences. Figure 3(b) and Table IV report the top five interesting patterns of length 3 that have been found by the MG-FSM algorithm. In particular, the sequence {*Colosseum* $\rightarrow$ *Roman Forum* $\rightarrow$ *St. Peter's Basilica*} is the most frequent route among places in Rome, followed by 4.4% of users.

C. Scalability evaluation

The goal of this evaluation is to assess the scalability of the ParSoDA applications discussed above, by analyzing their turnaround time by varying the number of computing

(a) Places of interest in Rome.



(b) Top 5 sequential patterns of lenght 3 in Rome.

Fig. 3. Sequential pattern mining application.

TABLE IV
TOP 5 SEQUENTIAL PATTERNS OF LENGTH 3 ACROSS PLACES IN ROME

Sequential pattern	Support
Colosseum → Roman Forum → St. Peter's Basilica	4.4%
Vatican Museums → St. Peter's Basilica → Colosseum	3.9%
Colosseum → Trevi Fountain → St. Peter's Basilica	3.7%
Colosseum → Roman Forum → Pantheon	3.6%
Colosseum → Pantheon → St. Peter's Basilica	3.6%

nodes involved in the parallel execution. Here we present the scalability results obtained with the sequential pattern mining application. The performance obtained with the frequent itemset applications are almost identical.

The scalability was evaluated running the data analysis applications on a Cloud platform. Specifically, we used *HDInsight*, a service that deploys an Apache Hadoop [17] cluster on Microsoft Azure[4]. Our cluster was equipped with 2 head nodes having four 2.2 GHz CPU cores and 14 GB of memory, and 12 worker nodes having four 2.2 GHz CPU cores and 14 GB of memory.

As shown in Figure 4), the turnaround time decreases from about 54 minutes using two workers, to 10 minutes using 12 workers. Thus, increasing the workers from 2 to 4 (2x), the obtained speedup is 1.98, and it is equal to 5.37 using 12 (6x) workers, with a speedup that is very close to linear values (see Figure 5). This behavior shows that the scalability of the ParSoDA applications was able to exploit the high-performance features of the Cloud platform and obtain a high level of scalability.

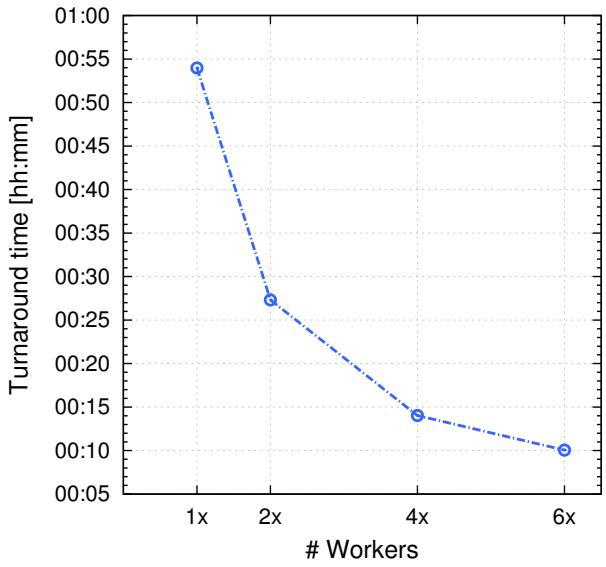


Fig. 4. Turnaround time of the sequential pattern mining application.

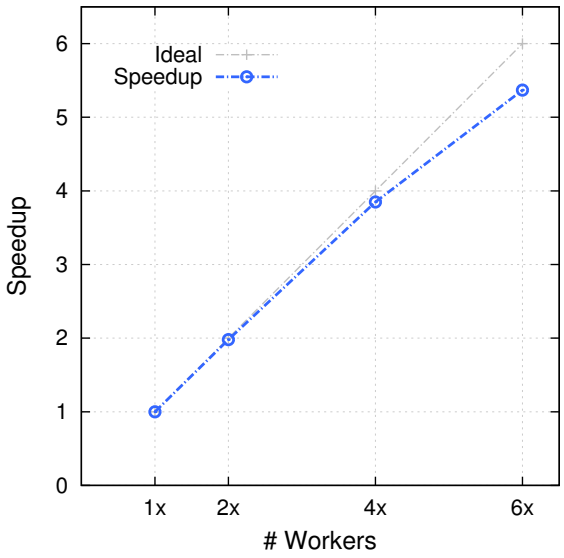


Fig. 5. Relative speedup of the sequential pattern mining application.

V. CONCLUSIONS

Social media analysis is an important research area aimed at extracting useful information from the big amount of data

[4]http://azure.microsoft.com/en-us/services/hdinsight

gathered from social networks. To cope with the size and complexity of social media data, the use of parallel and distributed data analysis techniques is fundamental. In this paper we presented ParSoDA, a Java library that can be used for building parallel social data analysis applications. ParSoDA was designed with the primary goal of reducing the programming skills necessary to implement the desired data analysis application.

ParSoDA defines a general structure for a social data analysis application that includes a number of steps (data acquisition, filtering, mapping, partitioning, reduction, analysis, and visualization), and provides a predefined (but extensible) set of functions for each step. Thus, an application developed with ParSoDA is expressed by a concise code that specifies the functions invoked at each step. In this way, even data analysts having limited programming skills, especially with regard to parallel programming, can efficiently design and execute data analysis applications dealing with big amounts of social media data.

To assess the scalability and the usability of ParSoDA, we developed two social data analysis applications to extract frequent itemsets and sequential patterns from social media data gathered from Flickr and Twitter that refers to the city of Rome. The experimental performance results presented in the paper showed that the scalability of the application using ParSoDA is able to exploit the high-performance features of the Cloud platform and obtain high level of scalability, with a global speedup that is very close to linear values.

REFERENCES

[1] Domenico Talia, Paolo Trunfio, and Fabrizio Marozzo. *Data Analysis in the Cloud*. Elsevier, October 2015.

[2] Bo Pang and Lillian Lee. Opinion mining and sentiment analysis. *Foundations and Trends in Information Retrieval*, 2(12):1–135, 2008.

[3] Eugenio Cesario, Andrea Raffaele Iannazzo, Fabrizio Marozzo, Fabrizio Morello, Gianni Riotta, Alessandra Spada, Domenico Talia, and Paolo Trunfio. Analyzing social media data to discover mobility patterns at expo 2015: Methodology and results. In *The 2016 International Conference on High Performance Computing and Simulation (HPCS 2016)*, Innsbruck, Austria, 18-22 July 2016. To appear.

[4] Eugenio Cesario, Chiara Congedo, Fabrizio Marozzo, Gianni Riotta, Alessandra Spada, Domenico Talia, Paolo Trunfio, and Carlo Turri. Following soccer fans from geotagged tweets at fifa world cup 2014. In *Proc. of the 2nd IEEE Conference on Spatial Data Mining and Geographical Knowledge Services*, pages 33–38, Fuzhou, China, July 2015. ISBN 978-1- 4799-7748-2.

[5] Nick Anstead and Ben O'Loughlin. Social media analysis and public opinion: The 2010 uk general election. *Journal of Computer-Mediated Communication*, 20(2):204–220, 2015.

[6] Abdul Ghaffar Shoro and Tariq Rahim Soomro. Big data analysis: Apache spark perspective. *Global Journal of Computer Science and Technology*, 15(1), 2015.

[7] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, OSDI'04, pages 10–10, Berkeley, USA, 2004.

[8] F. Marozzo, D. Talia, and P. Trunfio. A cloud framework for big data analytics workflows on azure. *Advances in Parallel Computing*, 23:182–191, 2013.

[9] J. Parthasarathi, K. Sundararaman, and G. S. V. Rao. Perisikan: An intelligent framework for social network data analysis. In *2012 International Conference on Communications and Information Technology (ICCIT)*, pages 13–16, June 2012.

[10] Sihem Amer-Yahia, Noha Ibrahim, Christiane Kamdem Kengne, Federico Ulliana, and Marie-Christine Rousset. Socle: Towards a framework for data preparation in social applications. *Ingénierie des Systèmes d'Information*, 19(3):49–72, 2014.

[11] Álvaro Cuesta, David F. Barrero, and María D. R-Moreno. A framework for massive twitter data extraction and analysis. *Malaysian Journal of Computer Science*, 27:1, 2014.

[12] Kristina Chodorow. *MongoDB: the definitive guide*. " O'Reilly Media, Inc.", 2013.

[13] Abid Hussain and Ravi Vatrapu. *Social Data Analytics Tool (SODATO)*, pages 368–372. Springer International Publishing, Cham, 2014.

[14] Stefan Stieglitz and Linh Dang-Xuan. Social media and political communication: a social media analytics framework. *Social Network Analysis and Mining*, 3(4):1277–1291, 2013.

[15] L. You, G. Motta, D. Sacco, and T. Ma. Social data analysis framework in cloud and mobility analyzer for smarter cities. In *Proceedings of 2014 IEEE International Conference on Service Operations and Logistics, and Informatics*, pages 96–101, Oct 2014.

[16] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In *Mass storage systems and technologies (MSST), 2010 IEEE 26th symposium on*, pages 1–10. IEEE, 2010.

[17] Tom White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 1st edition, 2009.

[18] Cheng Chu, Sang Kyun Kim, Yi-An Lin, YuanYuan Yu, Gary Bradski, Andrew Y Ng, and Kunle Olukotun. Map-reduce for machine learning on multicore. *Advances in neural information processing systems*, 19:281, 2007.

[19] ECMA. Ecma-262: ECMAscript Language Specification. Fifth edition. *ECMA (European Association for Standardizing Information and Communication Systems)*, 2009.

[20] Tom White. *Hadoop: The definitive guide*. " O'Reilly Media, Inc.", 2012.

[21] Iris Miliaraki, Klaus Berberich, Rainer Gemulla, and Spyros Zoupanos. Mind the gap: Large-scale frequent sequence mining. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 797–808. ACM, 2013.

[22] Jiawei Han, Jian Pei, Yiwen Yin, and Runying Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data mining and knowledge discovery*, 8(1):53–87, 2004.

[23] Haoyuan Li, Yi Wang, Dong Zhang, Ming Zhang, and Edward Y. Chang. Pfp: Parallel fp-growth for query recommendation. In *Proceedings of the 2008 ACM Conference on Recommender Systems*, RecSys '08, pages 107–114, New York, NY, USA, 2008. ACM.