Trajectory Pattern Mining for Urban Computing in the Cloud

Albino Altomare, Eugenio Cesario, Carmela Comito, Fabrizio Marozzo and Domenico Talia, *Member, IEEE*

Abstract—The increasing pervasiveness of mobile devices along with the use of technologies like GPS, Wifi networks, RFID, and sensors, allows for the collections of large amounts of movement data. This amount of data can be analyzed to extract descriptive and predictive models that can be properly exploited to improve urban life. From a technological viewpoint, Cloud computing can play an essential role by helping city administrators to quickly acquire new capabilities and reducing initial capital costs by means of a comprehensive pay-as-you-go solution. This paper presents a workflow-based parallel approach for discovering patterns and rules from trajectory data, in a Cloud-based framework. Experimental evaluation has been carried out on both real-world and synthetic trajectory data, up to one million of trajectories. The results show that, due to the high complexity and large volumes of data involved in the application scenario, the trajectory pattern mining process takes advantage from the scalable execution environment offered by a Cloud architecture in terms of both execution time, speed-up and scale-up.

Index Terms—Trajectory Mining, Urban Computing, Cloud Computing, Distributed Data Mining.

1 Introduction

1.1 Motivations

Urban computing [1] is the process of acquisition, integration, and analysis of big and heterogeneous urban data to tackle the major issues that cities face today, including air pollution, energy consumption, traffic flows, human mobility, environmental preservation, commercial activities and savings in public spending.

From a technological viewpoint, Cloud computing can play an essential role by helping city administrators to quickly acquire new capabilities and reducing initial capital costs by means of a comprehensive pay-as-you-go solution. In fact, by providing applications, infrastructure, networking, systems software, middleware and maintenance, Cloud computing lowers the barrier of entry and enables city managers to deliver high quality services to their citizens [2], [3], [4]. Working on such a research activity, we recently developed a Cloud-based framework specifically designed for developing urban computing solutions for smart cities [5]. The framework includes software layers for data management, service composition and application execution, integrated in a Cloud platform that interacts with data source generators like sensors, smart phones and other wireless devices. The framework includes a set of services allowing users to gather, collect, process, and analyze urban data in order to model social and environmental behaviors.

- *A. Altomare, E. Cesario and C. Comito are with the Institute of High Performance Computing and Networks (ICAR-CNR). Address: Via Bucci 41c, 87036 Rende (CS), Italy. E-mail: {altomare,cesario,comito}@icar.cnr.it.*
- *F. Marozzo and D. Talia are with the DIMES Department, University of Calabria, Rende (CS), Italy. E-mail: {marozzo,talia}@dimes.unical.it.*

1.2 Objectives and Contributions

The discovery of mobility models is one of the most challenging issues in urban computing. It can improve resource furnishing and management of cities [6], [7], [8]. For example, data generated by using mobile devices produce movement patterns that can be used to support the decisions of city managers in transport planning, intelligent traffic management, route recommendations, etc.

This paper describes the design and implementation of a parallel data mining methodology for discovering patterns and rules from trajectory data, performed over the Cloud-based framework presented in [9]. In particular, the applicative scenario described here focuses on the study of the trajectories followed by vehicles in an urban area with the aim to discover knowledge models, and thus to catch users' mobility behaviours. To this aim, the proposed algorithm is based on a two-steps approach: first, it detects dense regions within a given geographical area, i.e. more densely passed through regions, and then extracts trajectory patterns from those regions. The paper describes in details the design of the workflow implementing the application and its execution by a workflow engine. We apply the trajectory pattern extraction methodology to both real-world dataset (concerning mobility of citizens within an urban area) and synthetic datasets (built by a trajectory data generator). Experimental evaluation, carried out on a public Cloud platform, shows that, due to the high complexity and large volumes of data involved in the application scenario, the trajectory pattern mining process takes advantage from scalable execution offered by the Cloud. The results of a complete experimental evaluation point out advantages in terms of execution time, speed-up and scale-up.

For the sake of clarity, this paper extends the work presented in [9] and it provides several original contributions with respect to the previous one, as summarized in the following. First, the trajectory pattern workflow has been

enhanced, by exploiting an additional parallelism source aimed at improving the performance of the approach. In addition, the experimental testbed has been extended from 64 up to 128 servers, to perform a scalability analysis on a higher number of nodes. Moreover, the experimental evaluation has been performed also on synthetic datasets (other than a real-world dataset), by extending the size of input data from 80 thousands to 1 million trajectories, in order to carry out a scalability performance analysis on large datasets. Finally, an extensive experimental analysis has been devoted to investigate how the system performance, in terms of execution time and speed-up, is related to the number of computing nodes, the dataset size and dimension, as well as noise degree inside data.

Achieved results confirm the feasibility of the approach, the scalability and efficiency of the framework, and also show that it may be possible to optimize the performance by choosing the appropriate system and network configuration, for example, by tuning the number of workers and the number of data partitions. It is worth noting that the Cloud system allows developers and users to analyze large amount of trajectory data whose size is much higher than that can be analyzed by the most of the systems found in literature.

1.3 Plan of the paper

The rest of the paper is organized as follows. Section 2 outlines related work in the area of mobility pattern discovery, with a particular focus on Cloud-based urban computing. Section 3 reports the description of the Cloud-based framework and the facilities we used for the design and execution of data analysis workflows on the Cloud. Section 4 presents the trajectory analysis scenario describing the trajectory pattern detection methodology together with the designed workflow. Experimental evaluation both on a real use case scenario and synthetic datasets is reported in Section 5. Section 6 concludes the paper and plans further research works.

2 Related Work

As mentioned before, the objective of this work is twofold: (i) implementing a Cloud-based software environment for efficiently manage socio-environmental data, with a particular focus on the urban context of cities, and (ii) defining a methodology to analyze trajectories of mobile users in order to mine social and environmental behaviors. Accordingly, in this section we will briefly review some of the most representative research work in both the areas.

2.1 Cloud-based Urban Computing

Several Cloud enabled tools for urban planning and management in the smart city context have been recently proposed. Environmental Software and Services (ESS) [2] exploits the Cloud paradigm to offer a range of services for environmental planning and management, policy and decision making, world wide. Analogously, the Environmental Virtual Observatory pilot (EVOp) [3] uses Clouds to achieve similar objectives in the soil and water domains.

The European Platform for Intelligent Cities (EPIC) [10] combines a Cloud computing infrastructure with the knowledge and expertise of the Living Lab approach to deliver sustainable, user-driven web services for citizens and businesses.

The Life 2.0 project [11] offers a set of services ranging from basic geographical positioning systems to socially networked services and to local market-based services. The project aims to provide solutions that increase opportunities for social contacts among elderly people in their local area, by providing new services for elderly people, based on the use of tracking systems and social network applications.

IBM introduced Smarter City Solutions on the IBM SmartCloud Enterprise, a public Cloud platform that includes hardware, network and storage [4]. The platform provides pay-as-you-go services for urban management within cities. Those services include application software, infrastructure, networking, systems software, middleware and maintenance.

2.2 Trajectory Pattern Mining

Discovering periodic patterns from historical object movements is a very challenging task. In [12] an approach to discovery hidden periodic patterns in spatio-temporal data is proposed. In particular, authors define the spatio-temporal periodic pattern mining problem and propose an algorithm for retrieving maximal periodic patterns. Moreover, they devise a specialized index structure, aimed at supporting more efficient execution of spatiotemporal queries over the discovered patterns.

A prediction approach to estimate an object future location, based on its pattern information and recent movements, is proposed in [13]. Specifically, the discovered trajectory patterns are stored in the TPT, a tree data structure exploited for an efficient and accurate prediction of future locations. In addition, two query processing techniques are presented, to perform both near and distant time predictive queries on the TPT structure.

Reference [14] presents a smart driving direction system, where GPS-equipped taxis are employed as mobile sensors aimed at probing the traffic rhythm of a city. In particular, the main idea is to exploit the intelligence of experienced taxi drivers so as to provide a user with the practically fastest route to a given destination at a given departure time. The system has been tested on a real-world trajectory dataset generated by over thirty thousands taxis in a period of 3 months, aimed at evaluating the effectiveness of the approach.

In [15] authors extend the sequential pattern mining methodology to analyze moving objects. Some approaches of different complexity are proposed, that have been empirically evaluated over real data and synthetic benchmarks, comparing their strengths and weaknesses.

Differently from the approaches described above, at the best of our knowledge, our work is novel in two aspects:

1) It is a pioneering workflow-based approach to mine trajectory patterns on a real public Cloud platform. This allows us to provide an integrated methodology for efficiently manage trajectory data where

given as input a GPS dataset, the methodology directly produces as output a set of dense regions and trajectory patterns for the specific dataset. Moreover, the Cloud allows us to analyze large amounts of trajectory data whose size is much higher than that can be analyzed by the most of the systems found in literature.

2) The experimental tests performed on these large datasets show that the whole process takes advantages from such a scalable environment, both in terms of execution time and achieved speed-up.

Moreover, another important distinguishing feature of our system is that it has been tailored to provide general-purpose services for urban planning and management within the city context. Nevertheless, the system has been designed as a set of modular components allowing easy extensibility and integration of different heterogeneous components (e.g., software, data sources, etc).

3 A Cloud-based Environment for Urban Computing

In this section we introduce the architecture of the software environment we developed for the implementation of services aiming at improving the planning, managing and monitoring of activities within a urban context, such as healthcare, smart transportation, smart home, smart tourism and smart public services. The proposed architecture has been designed as a middleware layer supporting the integration and handling of large-scale, fragmented, cross-thematic environmental and socio-geographic data with the focus of mining human behavior from such data for urban planning and management. The Cloud computing paradigm allows the implementation of the above urban-related services by facilitating data access and storage across platforms, providing on-demand computational resources, and allowing for integrated processing and data analysis.

Figure 1 shows the architecture consisting of a set of modular layers. At the lower level, the *Platform layer* is based on a hybrid Cloud environment that ensures cross-platform accessibility of environmental data. This layer can be made more efficient and functional by integrating other systems as MapReduce, Storm and Kafka. The *Data Acquisition layer* allows accessing environmental data collected from disparate sources, to monitor urban services such as water quality, energy usage, etc. At the *Data Storage level* the data collected is organized in ad-hoc repositories (i.e., historical archives and real-time repositories). The *Software Service layer* is composed of a set of software components exposed as services, that can use data provided by the lower level and are invoked by the upper level to compose applications. The *Service Composition layer* is responsible to assist users in designing application workflows, identify data sources, and link necessary processing components to enact the workflows. Finally, the *Smart Urban Application Services layer* offers a set of services for urban management, that can be used to perform intelligent analysis on environmental data. For lack of space, no more details are reported in this paper. A list of the main functionalities of the framework can be found in [5]. The software architecture has been developed and deployed on a public Cloud for internal use. Although it is not available on-line for public access, it can be released for research purposes to academic or research institutions.

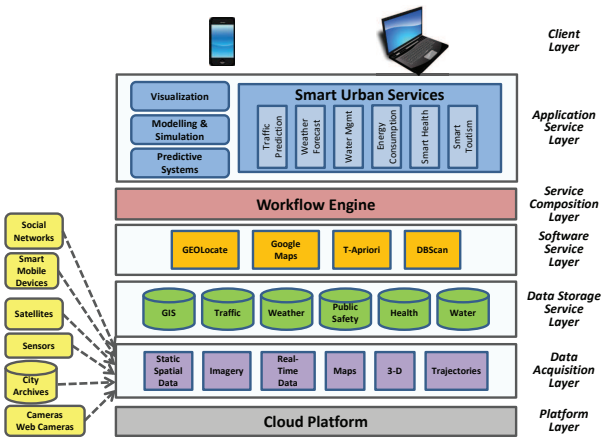


Fig. 1. A Cloud-based architecture for urban computing.

The implementation of the Service Composition Layer has been done using the *Data Mining Cloud Framework* (DMCF) [16], a software framework for designing and executing data analysis workflows on the Cloud. DMCF supports a large variety of processing patterns that can be used in data mining, including single-task applications, parameter sweeping applications, and workflow-based applications. Following the approach proposed in [17], DMCF represents knowledge discovery workflows as graphs whose nodes denote resources (datasets, data analysis tools, mining models) and whose edges denote dependencies among resources. A Web-based user interface allows users to compose their workflows and to submit them for execution to the Cloud platform, following a Software-as-a-Service (SaaS) approach. Data analysis workflows can be designed through visual programming, which is a very effective design approach for high-level users, or through a script-based language [18], which is an additional and more flexible programming interface for skilled users.

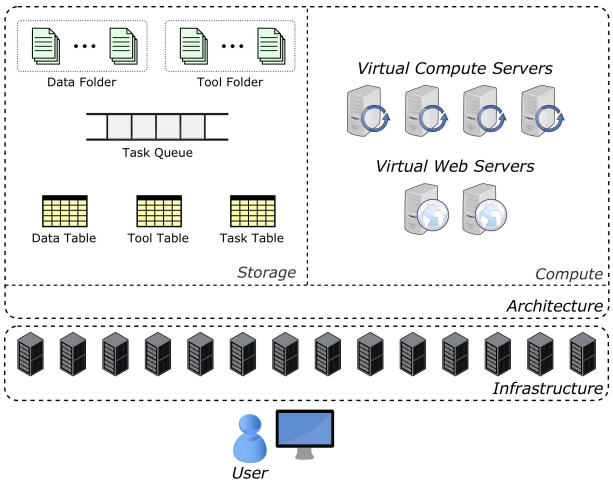


Fig. 2. Architecture of the Data Mining Cloud Framework.

Figure 2 shows the architecture of the DMCF that includes different kinds of components grouped into storage

and compute components. The storage components include:

- A *Data Folder* that contains data sources and the results of knowledge discovery processes. Similarly, a *Tool Folder* contains libraries and executable files for data selection, pre-processing, transformation, data mining, and results evaluation.
- The *Task Queue* contains the tasks ready for execution.
- *Data Table*, *Tool Table* and *Task Table* contain metadata information associated with data, tools, and tasks.

The compute components are:

- A pool of *Virtual Compute Servers*, which are in charge of executing the data analysis tasks.
- A pool of *Virtual Web Servers* host the Web-based user interface.

The following steps are performed to develop and execute a knowledge discovery application [19]:

1) A user accesses the Website and designs the application (either single-task, parameter sweeping, or workflow-based) through a Web-based interface.
2) After application submission, the system creates a set of tasks and inserts them into the Task Queue on the basis of the application.
3) Each idle Virtual Compute Server picks a task from the Task Queue, and concurrently executes it.
4) Each Virtual Compute Server gets the input dataset from the location specified by the application. To this end, a file transfer is performed from the Data Folder where the dataset is located, to the local storage of the Virtual Compute Server.
5) After task completion, each Virtual Compute Server puts the result on the Data Folder.
6) The Website notifies the user as soon as her/his task(s) have completed, and allows her/him to access the results.

The set of tasks created on the second step depends on how many data analysis tools are invoked within the workflow. Initially, only the workflow tasks without dependencies are inserted into the Task Queue. All the potential parallelism of the workflow is exploited by using the needed Virtual Compute Servers. In addition, multithreaded tasks exploit all the cores available on the Virtual Compute Servers they are assigned to.

The DMCF architecture has been designed in a sufficiently abstract and generic way to be implemented on top of different Cloud systems. In this work, we use the implementation based on Microsoft Azure[1].

To reduce the overhead of data transfers between Data Folder and the local storage of Virtual Compute Servers, it is important that data are kept physically close to the virtual servers where processing takes place. In the Microsoft Azure implementation, this is achieved by exploiting the Azure's Affinity Group feature, which allows Data Folder and Virtual Compute Servers to be located near to each other in the same data center for optimal performance.

1. http://azure.microsoft.com

4 The Trajectory Pattern Mining Methodology

This section provides a real-world application scenario as a case study of urban planning and management within the proposed framework. In particular, we focused on the study of the trajectories traced by vehicles or humans, with the aim to discover user's behavior and provide useful information about mobility-related phenomena. To this aim, we propose a trajectory pattern extraction methodology allowing to predict future movements of citizens, in order to support decisions in urban contexts. The set of trajectory patterns extracted represent a basic building block around which further tasks can be implemented, including the following ones:

- *Next location prediction.* Predict the future location of a moving object, based on the object recent movements and trajectory pattern models, to anticipate or prefetch possible services in that location.
- *Intelligent traffic management.* Predict traffic congestion patterns and improve the transportation model of a city, to reduce the wasted time due to vehicular traffic.
- *Movement-similarity analysis.* Estimate the similarity between users in terms of location histories so as to promote services for car sharing, car pooling, etc.
- *Travel recommendations.* Mine the top interesting locations and travel sequences among locations, and exploit such information to recommend the best routes and itineraries that people can follow to visit a given location.

In this section we first describe the trajectory pattern extraction methodology to analyze routes drawn by users during their daily activities. Then, we point out how a workflow mechanism can be used to design the methodology within a parallel setting, as the one of the proposed Cloud-based architecture (see Figure 1).

4.1 Trajectory Pattern Detection Approach

Before describing the approach, let us introduce some notation used in the remainder of the section. Let be $T =< t_1, t_2, \ldots, t_H >$ an ordered timestamp list, such that $t_h < t_{h+1}, \forall_{0<h<H}$. A *raw trajectory* (or simply *trajectory*) τ_K is a spatio-temporal sequence

$$\tau_K =< (x_{1K}, y_{1K}, t_1), \ldots, (x_{HK}, y_{HK}, t_H) >$$

where each triple (x_{iK}, y_{iK}, t_i) indicates that an object of the trajectory τ_K is in the position (x_{iK}, y_{iK}) at time t_i. The *trajectory length* is the number of triples composing the trajectory (i.e., $|T| = H$). A *frequent (or dense) region* is an area of points that is more frequently visited by the object's trajectories with respect to other areas. In particular, we represent with R_t^j the j^{th} dense region at the time t. A *structured trajectory* τ_K is a spatio-temporal sequence, $\tau_K =< R_{t_1}^{j_1}, \ldots, R_{t_H}^{j_H} >$, where each element $R_{t_i}^{j_i}$ indicates that an object of the trajectory τ_K is in the dense region R^{j_i} at time t_i. A *trajectory pattern* is a special association rule, in the form

$$R_{t_1}^{j_1} \wedge R_{t_2}^{j_2} \wedge \ldots \wedge R_{t_r}^{j_r} \xrightarrow{c} R_{t_s}^{j_s}$$

with time constraints $t_1 < t_2 < \ldots < t_r < t_s$. The block on the left, i.e. $R^{j_1}_{t_1} \wedge R^{j_2}_{t_2} \wedge \ldots \wedge R^{j_r}_{t_r}$ is the primes, while $R^{j_s}_{t_s}$ is the consequence of the rule. Finally, c is the confidence of the rule, meaning that when the premise occurs then the consequence will occur with probability c.

Now, let us describe the approach adopted to detect trajectory patterns, which is composed of three main steps. To better describe the whole process, Figure 3 shows a graphic representation of how trajectory patterns are discovered. The input data of the analysis is a set of raw trajectories, that have been obtained by sampling real trajectories traced by users during their daily activities. The first step of the algorithm consists in the *detection of frequent regions* from the original raw trajectory dataset. The goal of this step is detecting spatial areas more densely passed through, in order to conduct the further analysis as movements through areas rather than single points. The second step consists in the *synthesization of the trajectories*, by changing their representation from movements between points into movements between frequent regions. Precisely, each point of the original dataset is substituted by the region it belongs to. The third step is aimed at *extracting trajectory patterns*, in the form of associative rules, analyzing the trajectories of frequent regions obtained at the previous step.

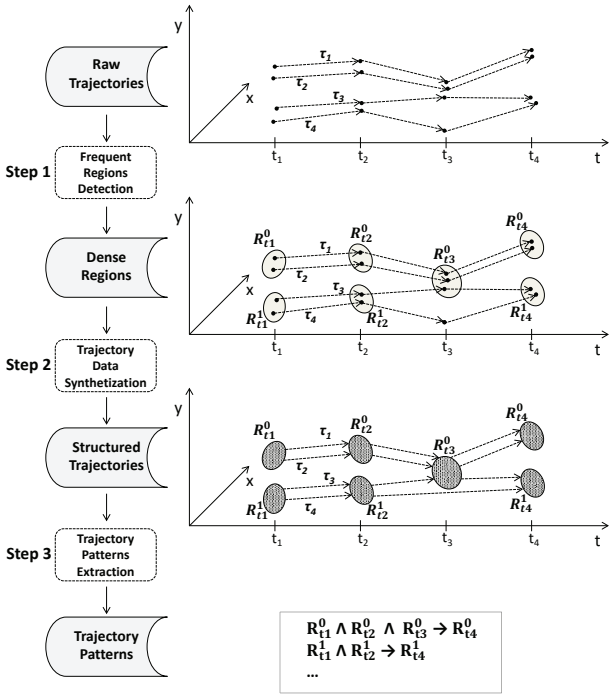


Fig. 3. Trajectory Pattern Detection Steps.

4.2 Trajectory Pattern Detection: a parallel implementation

The trajectory pattern detection process consists of a sequence of steps involving different kinds of data and tools that can be located over geographically distributed environments. Moreover, some steps can be naturally parallelized, in order to achieve higher performance . In particular, as it will be better shown in the experimental evaluation section, the frequent regions detection and trajectory synthesization steps are the most time-consuming and critical tasks. Thus, our effort consists in the parallelization of this step (that has been done by implementing a Single Program Multiple Data parallelism pattern).

Now, in order to have a clear view of the whole process, Figure 4 shows it by exploiting the workflow formalism, i.e. a graph in which nodes represent data sources, data mining tools and algorithms, and edges represent execution dependencies among nodes. The original data set D is a raw trajectory data, populated by the trajectories (represented in the previously described format) of persons collected somehow. In particular, let us suppose that the original dataset is composed of N trajectories, each one represented as a sequence of H (x, y, t)-triples.

The workflow is composed of four steps (see Figure 4), as described in the following:

Step 0 - Vertical/Horizontal Data Splitting. The original trajectory dataset is partitioned by the *Time Stamp Splitter* in a vertical way, with respect to the timestamp value. In other words, the points of the trajectories visited at the time stamp $t_i \in T$ will be gathered in the i^{th} output dataset, for $i = 1, ..., H$. At the end of this step, H different datasets are available, each one containing a vertical projection of D on the timestamp t_i. It is worth noticing that this is an additional step with respect to the sequential case, where no splitting step is planned. In parallel to the vertical splitting, the trajectory dataset is horizontally divided in M partitions by the *Trajectory Splitter*. Each partition $D_1, ..., D_M$ contains a subset of trajectory in D, where $|D_i| \cong \frac{1}{M}|D|$, for $i = 1, ..., M$. Such data partitioning is aimed at improving the scalability of the synthesization step.

Step 1 - Frequent Regions Detection. This step is aimed at detecting, for each timestamp, the regions that are more densely visited with respect to others (thus, of interest for the further analysis). In the workflow this is done by running H clustering algorithm instances, each one taking in input a dataset built at the previous step. The final result consists of H clustering models, whereas the clusters of the t_h-model represent the detected dense regions of the t_h-timestamp (each cluster corresponds to a dense region). The number of detected regions (i.e., number of clusters) may be different for each timestamp t_h.

Step 2 - Trajectory Data Synthesization. This step is aimed at synthesizing the trajectories to build a structured trajectory dataset. This task is performed by running the *Trajectory Synthesizer* tool, whose goal is to create a dataset where each point of the original trajectories is substituted by the dense region it belongs to (discovered at the Step 1). The final dataset, the *Trajectory Data (structured)* in the figure, is populated by trajectories between dense regions (but between single points).

Step 3 - Trajectory Pattern Extraction. Finally, a *Trajectory Pattern Extraction* algorithm on the dense regions trajectory data is executed, to discover trajectory patterns from them. The final mining model is a set of associative rules describing spatio-temporal relations between the movement of the users under investigation.

5 Experimental Evaluation

To evaluate the performance and the effectiveness of the system that has been described in the paper, we carried out

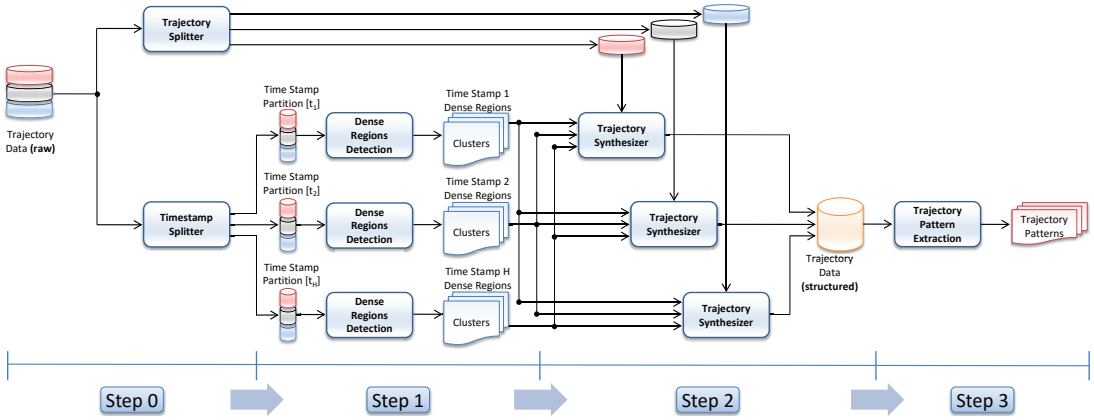


Fig. 4. Trajectory Pattern Detection Workflow.

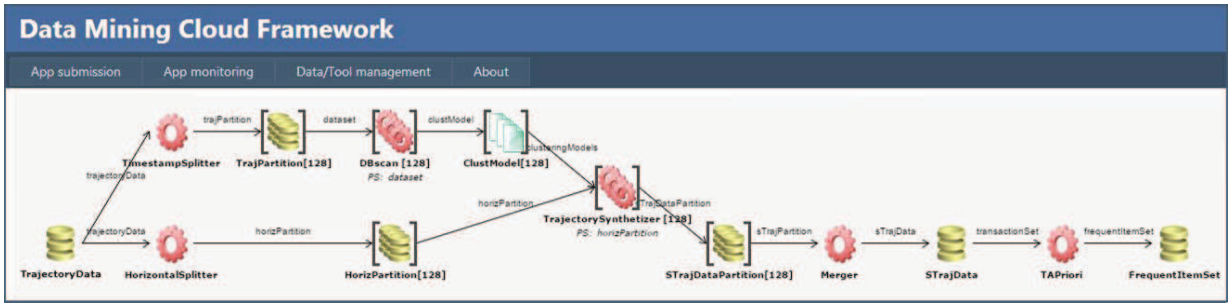


Fig. 5. Trajectories workflow composed and executed in the Data Mining Cloud Framework (DMCF)

an extensive analysis by executing different experiments in various scenarios. Since the proposed algorithm is specifically designed to deal with large dataset of high cardinality (number of trajectories) and dimensionality (number of timestamps), it is important to measure its performance on the basis of such parameters. Also, notice that, since the computational complexity cannot be analytically devised, experimental performance analysis is critical in the approach. To this purpose, it is important to provide an efficient implementation of tools and algorithms that work effectively even in ”extreme” situations.

The trajectory pattern mining application has been executed on a real Cloud environment, exploiting the system described in Section 3. The whole process, designed as a workflow, has been composed and run by the Data Mining Cloud Framework exploiting up to 128 servers. More details about experimental setting and workflow description are reported in Section 5.1. Experiments have been carried on both real and synthetic data, and their results are cross validated across different orders of the data. Section 5.2 presents the results about the efficiency of the approach carried out on a real-life data set, as well as some example patterns discovered in a real case. In particular, the algorithm has been tested on T-Drive, a GPS dataset tracing the movement of taxies in the urban area of Beijing. Also, several synthetic data sets are exploited in Section 5.3 to investigate scalability and robustness in critical applicative settings. Such data sets have been built by an ad-hoc trajectory data generator that we designed and implemented.

5.1 Cloud Experimental Setting and Applicative Workflow

The trajectory pattern mining application has been developed and executed on the Cloud System described in Section 3, exploiting the Data Mining Cloud Framework. We executed the experiments on the Microsoft Azure platform using 1 virtual server to run the Data Mining Cloud Framework Website, and up to 128 virtual servers for the Workers. Each virtual server was equipped with a single-core 1.66 GHz CPU, 1.75 GB of memory, and 225 GB of disk space.

Figure 5 shows a snapshot of the workflow designed through the Service Composition Layer. Each node represents either a data source or a data mining tool, whereas an edge represents an execution dependency among nodes. For what concerns the algorithms, the *Frequent Regions Detection* step has been implemented by using DBScan [20], a density-based clustering algorithm, whereas the *Trajectory Pattern Extraction* step has been performed by T-Apriori, our ad-hoc modified version of the well-known Apriori algorithm [21]. Moreover, some nodes are labeled by the array notation, which is a compact way to represent multiple instances of the same dataset or tool. For example, the *”DBScan[128]”*-labeled node represents 128 parallel instances of the algorithm, each one belonging to a different path of the workflow.

The workflow shown in Figure 5 implements the trajectory analysis steps shown in Figure 4. The initial dataset, *Trajectory Data*, is partitioned into H (i.e.,$= |T|$) subsets using the *Time Stamp Splitter* tool, where H is equal to the number of timestamps (the points in the trajectory).

In the example shown in Figure 5, $H = 128$. This step, corresponding to the Step 0 of the workflow shown in Figure 4, produces H data partitions. Now, each partition $TrajPartition[i]$, $i = 1, ..., H$, is analyzed by an instance of $DBScan$ and produces a $ClusteringModel$ (Step 1). Each clustering model is a set of clusters/dense regions, for a given timestamp. The *TrajectorySynthetizer* tool analyzes all models and the initial dataset, so as to generate the *Structured Trajectory Data*, where each point of the original trajectories is substituted by the dense region it belongs to (Step 2). Finally, the *T-APriori* gets in input this dataset to extract trajectory patterns and, thus, produces the final results (Step 3).

Experimental tests have been executed on both real and synthetic data, having in mind two different goals. The experimental evaluation on real data has been performed to show a concrete scenario on which our approach can be applied on (it concerns mobility aspects of a big city like Beijing) and the practical usefulness of the system in real urban cases. With this aim, we will show the discovered dense regions and the most frequent mobility patterns traveled in the real city scenario. In order to perform a more complete scalability analysis (that we could not achieve on the real dataset due to its limited cardinality), we will test the system also on synthetic datasets to deal with higher orders of magnitude under different settings and with respect to several data sizes (up to one million trajectories).

The goal of the evaluation is to assess the execution time and scalability of the whole task, by analyzing the time elapsed in each step and comparing the performances obtained by both sequential and parallel executions. In particular, we evaluated the results by exploiting the following performance metrics:

- *Turnaround time*: the total execution time of the distributed algorithm varying the number of running nodes, that is, the elapsed time from task submission until the final result is returned to it;
- *Speed-up*: the ratio of the turnaround time elapsed by exploiting 1 node to the turnaround time on n nodes, which measures how much performance gain is achieved by parallelizing a given application over a sequential implementation;
- *Scale-up*: the execution time when the problem size is increased linearly with the number of servers, which quantifies the capability of a system to handle larger data sets when (computational) resources are added to accommodate that growth;
- *Efficiency*: the ratio between speedup and the number of processing nodes, which measures the percentage of time for which processing nodes are usefully exploited for computation (and not for communication tasks or even idling).

In the following we will describe the achieved results, obtained by an extensive evaluation carried out in various experimental scenarios.

5.2 Real-life Data

In this section we explore a trajectory analysis case study, by applying the pattern mining detection method described in the previous section over a real dataset. In particular, we report the results obtained by the execution of the methodology on T-Drive, a real-life GPS dataset tracing the movement of taxies in the urban area of Beijing. A detailed description of the dataset is reported in Section 5.2.1. The description of the analysis and the most important results (dense areas and mobility patterns) carried on T-Drive are reported in Section 5.2.2.

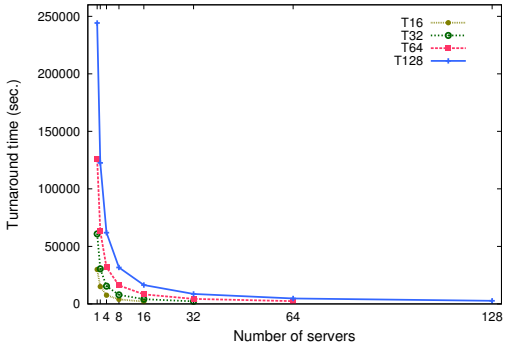
5.2.1 Data Description.

The input dataset chosen for the experiments is the *T-Drive Trajectory Data Sample* [22], [23], a collection of GPS traces describing the movement of GPS-equipped taxis in the urban area of Beijing, China. The temporal span of the dataset is one week. The number of vehicles tracked is 10,357. We have about 15 millions of locations (geographic points) and the total area covered by the trajectories reaches almost 9 million kilometers. Starting from this dataset, we extracted a subset of 80,000 trajectories, obtained by sampling taxi positions every 5 minutes. Then, from this dataset we created four different ones, all of 80,000 trajectories, that differentiate only for the length of the trajectories. In particular, we built datasets whose trajectories are traced by 16, 32, 64 and 128 samples (i.e., timestamps), referred in the following as $T16$, $T32$, $T64$ and $T128$, respectively. Those four datasets have been used in the experimental evaluation.

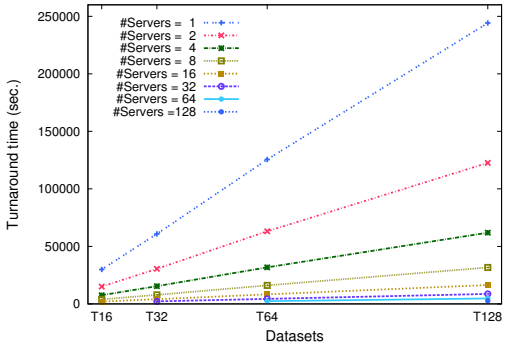
Before the analysis of the trajectories, a pre-processing has been performed to clean, select and transform data to make it suitable for analysis. First, we cleaned collected data by removing all the points with unreliable position (i.e., coordinates with latitude-longitude equals to 0.0 or 7.0 are evident mistakes). Then, to avoid any other kind of geo-localization errors, we selected only data points falling in a bounded area limiting the city, by removing points outside this area. Overall data errors amounted to about 0.7% of points. Finally, we transformed data by partitioning each trajectory in a daily route, because we were interested to discover daily patterns inside data. The final dataset contains about 61,500 daily trajectories, each one containing the set of points traced by a single taxy during a day. The total data size amounts to about 882 MB.

5.2.2 Experimental Results.

First, we measured the turnaround times of the application for the four considered datasets, using from 1 to 128 virtual servers. Figure 6 shows such results. Therefore, the shown plots can also be seen as a comparison between a parallel and a sequential solution. In particular, Figure 6(a) shows how the turnaround time decreases with higher number of virtual machines, for different dataset sizes. For instance, for the 16 timestamp dataset the turnaround time decreases from around 8.3 hours obtained on a single server, to about 34 minutes on 16 servers. For the 32 timestamp dataset the turnaround time diminishes from 17 hours to 38 minutes. For the 64 timestamp dataset the turnaround time decreases from 35 hours to 40 minutes. Finally, with the 128 timestamp, the turnaround time ranges from about 68 hours to about 45 minutes using 128 virtual machines. Figure 6(b) shows how the turnaround time increases with respect to the dataset size, for a different number of virtual machines. The graph shows that the time required to execute the entire

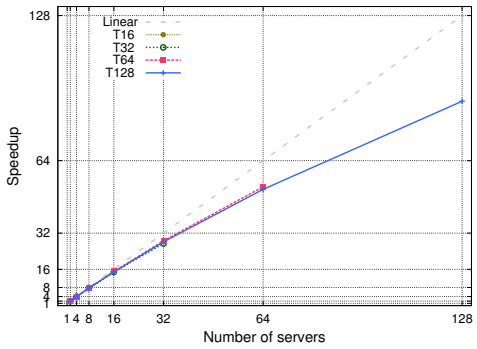


(a) Turnaround time vs the number of available servers, for different data sizes.

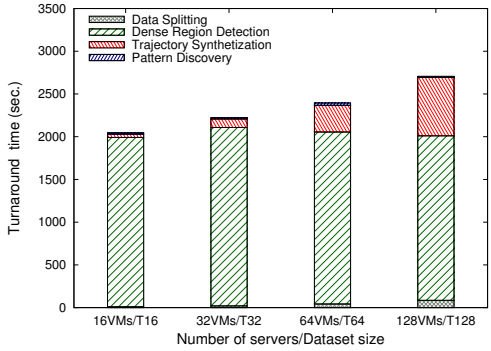


(b) Turnaround times vs data sizes, for different number of available servers.

Fig. 6. T-Drive: Turnaround times in different scenarios.



(a) Speed-up vs the number of available servers, for different dataset sizes.



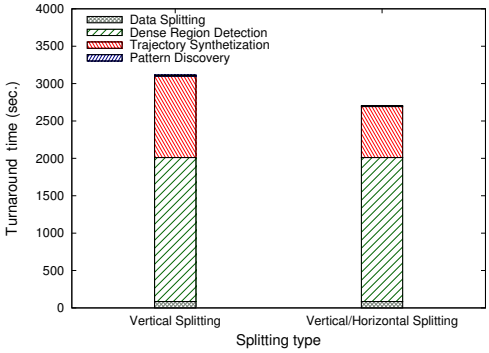
(b) Scale-up with the partial times required by each step of the workflow, for different dataset sizes.

Fig. 7. T-Drive: Speed-up and Scale-up.

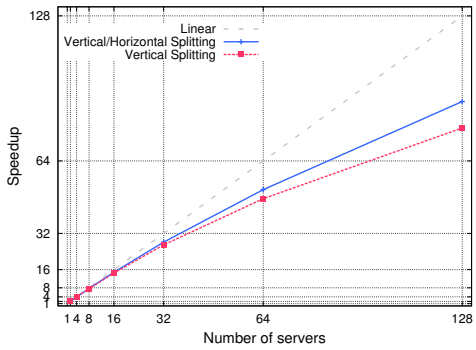
workflow increases proportionally with the increase of the input size. On the contrary, the time required to execute the entire workflow decreases proportionally with the increase of computing resources.

Figure 7 shows execution speed-up and scale-up values. More in detail, Figure 7(a) shows that the speed-up is linear with all datasets, up to the case of 16 nodes: this represents a good trend. For an higher number of nodes, the speed-up is not linear because of the influence of the sequential steps of the application. In fact, for the 32 timestamp dataset, it ranges from 2.0 using 2 servers, to 27.4 using 32 servers, while for the 64 timestamp dataset the speed-up ranges from 2.0 to 52.4. Finally, with the 128 timestamp it ranges from 2.0 to 78.5 by using 128 servers. Figure 7(b) measures the application scale-up by showing the turnaround times obtained when the size of the input dataset increases proportionally to the number of virtual servers exploited for the computation (i.e., 16 timestamps on 16 servers up to 128 timestamps on 128 servers). The results show that the total turnaround time is almost constant up to 64 timestamps on 64 servers and slightly increases in the case 128 timestamps/128 servers. This demonstrates that the amount of data that can be analyzed in a given amount of time increases, almost linearly, with the number of computing resources available. Other than showing the total turnaround time, Figure 7(b) shows the time required by each step of the workflow. We can notice that in each scenario the Dense Region Detection step takes most of the total time and this time is almost constant in all the four scenarios. This is due to the fact that the parallelization degree of clustering algorithm executions (i.e., DBScan) increases proportionally with the dataset size. On the other side, the time required by Data Splitting and Pattern Discovery steps, that are implemented as sequential tasks, increases for larger dimensions of the dataset. The same behaviour holds for the Trajectory Synthesization step, even if it has been parallelized: this last issue is better explained below.

Now, let us describe the influence of the vertical and horizontal data splitting on the whole execution time. Figure 8 shows the turnaround time and speed-up when processing the $T128$ dataset on 128 nodes. In detail, we can observe in Figure 8(a) that when only vertical splitting is done, the turnaround time amounts to 3115 seconds. Implementing both horizontal and vertical splitting decreases the turnaround time to 2707 seconds, that is obviously due to a reduction in the trajectory synthesization step. This improves also the speed-up trend, that increases from 78.5 to 90.2 (as can be observed in Figure 8(b)). Nevertheless, horizontal splitting benefits are not proportional to the its parallelism degree. The last consideration is better highlighted in Figure 9, which reports the execution time taken by the synthesizer versus the number of horizontal partitions, when processing the T128 dataset with 128 vertical partitions (and 128 running nodes). Specifically, the elapsed



(a) Turnaround times with the partial times required by each step of the workflow, with T128 and 128 partitions.



(b) Speed-up vs the number of available servers, with T128.

Fig. 8. T-Drive: horizontal/vertical splitting versus horizontal splitting.

time decreases with the the parallelism degree but till 8 horizontal partitions.

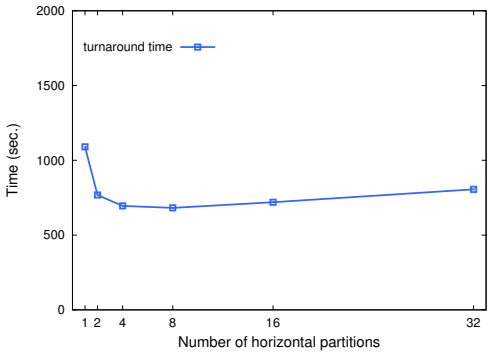


Fig. 9. T-Drive: Trajectory Synthesizer time vs number of horizontal partitions.

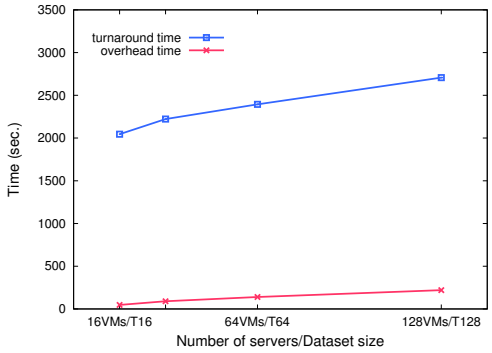
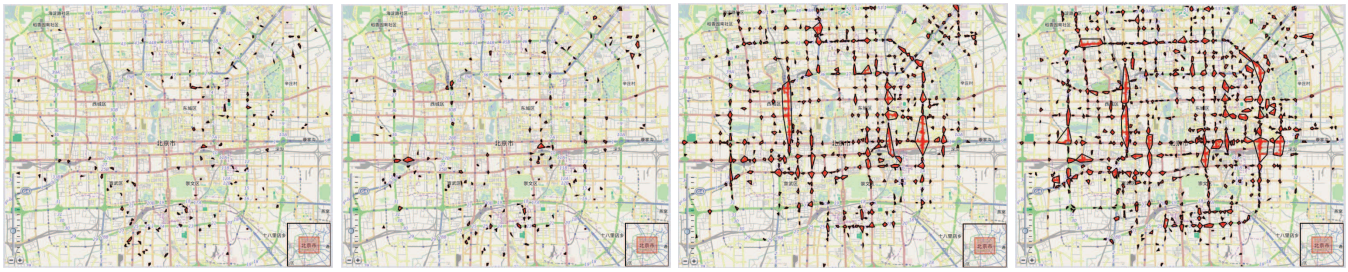


Fig. 10. T-Drive: Overhead time vs data sizes.

We evaluate also the overhead introduced by the Data Mining Cloud Framework. We defined as overhead the time required by the system to perform preliminary operations (e.g., getting the task from the Task Queue, downloading the libraries and the input from the Cloud storage) and final operations (e.g., update of the Task Table, upload of the output results in the Cloud storage) of each step of a workflow to execute. The overhead increases with the number of tasks and the size of the datasets involved in the computation. In fact, the more the tasks involved, the longer the time elapsed for their submission, monitoring, finalization, etc. Similarly, the higher the data size, the longer the cumulative time spent for its transfer. This trend is confirmed in Figure 10, that shows the turnaround and the overhead times, when the size of the input dataset increases proportionally to the number of virtual servers. We can observe that the overhead takes only a very small amount of the total turnaround time. For example, the overhead of the analysis of the 16 timestamp dataset takes 47 seconds on a total execution time of 2045 seconds, while the 128 timestamp dataset takes 220 seconds on a total of about 2707 seconds. This means that the system overhead is just the 2.3% and the 8.1% of the total execution, respectively.

Finally, through the visualization module of the proposed framework, we show some dense regions (i.e., the most congested areas of the city) discovered through the T-Drive dataset analysis. Figure 11 shows some examples, for different time windows (3 hours) of the day. Interestingly, such images show how the taxi mobility and traffic congestion changes over the morning and afternoon. For example, we can observe that, during the early morning (around 6 AM and 9 AM), a low number of dense regions have been discovered, the most of them localized in south and west areas of the city (Figures 11(a) and 11(b)). During the day, the distribution of vehicles increases and the traffic becomes more chaotic in several areas. In particular, we can observe (Figures 11(c) and 11(d)) that from the late morning to the afternoon many regions in the city have an high concentration of taxies. We can clearly recognize the main streets that are exploited during these times: several highways crossing the central area of the city, as well as a circular highway around the city center (clearly observable in Figures 11(d)) and an highway toward the airport. It is worth pointing out that the dense regions do not necessarily indicate traffic problems in those areas. These regions represent dense movement of cars, which can hint the possibility of traffic jams or congestions. Further analysis, focused on these specific areas, are needed to have a more precise indication of possible traffic problems.

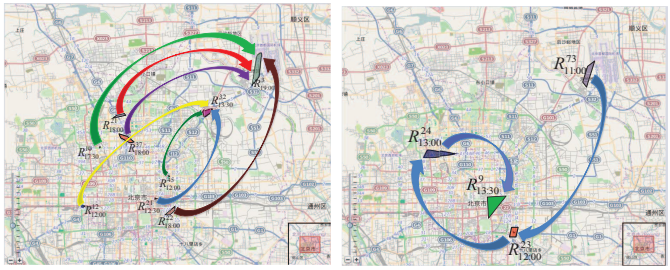
Figure 12 shows some examples of the most popular routes discovered in T-Drive by the TPM algorithm. To detect the most popular itineraries, we concentrate the



(a) time: 6:00 AM (b) time: 9:00 AM (c) time: 12:00 AM (d) time: 3:00 PM

Fig. 11. T-Drive: Dense regions discovered in T-Drive, w.r.t. several morning and afternoon time windows.

analysis on routes around the city center and those going from the city center to strategic venues like the airport and the train stations. Figure 12(a) highlights the main extracted mobility patterns followed by taxi drivers leaving the center toward airport (i.e., rules $R^{12}_{12:00} \rightarrow R^{32}_{13:30}$, $R^{21}_{12:30} \rightarrow R^{32}_{13:30}$, $R^{10}_{17:30} \rightarrow R^{3}_{19:00}$, $R^{22}_{18:00} \rightarrow R^{3}_{19:00}$, etc.). It is evident that vehicles departing from different locations in the city center ($R^{12}_{12:00}, R^{21}_{12:30}, R^{10}_{17:30}, R^{22}_{18:00}$, etc.) converge toward two regions (i.e., $R^{32}_{13:30}$ and $R^{3}_{19:00}$). On the other side, the pattern in Figure 12(b) shows a popular itinerary from the airport to a train station in the city center. The pattern is composed of 3 trips. The first one ($R^{73}_{11:00} \rightarrow R^{23}_{12:00}$) goes from the airport to a popular venue in the sub-urban south area of the city. The second trip ($R^{23}_{12:00} \rightarrow R^{24}_{13:00}$) goes from this sub-urban area to the city center and the last one ($R^{24}_{13:00} \rightarrow R^{9}_{13:30}$) from the city center to the train station. This pattern represents the most crowded route to get the central train station from the airport.



(a) From the city center to the airport. (b) From the airport to the train station.

Fig. 12. Travel patterns discovered in T-Drive.

5.3 Synthetic Data

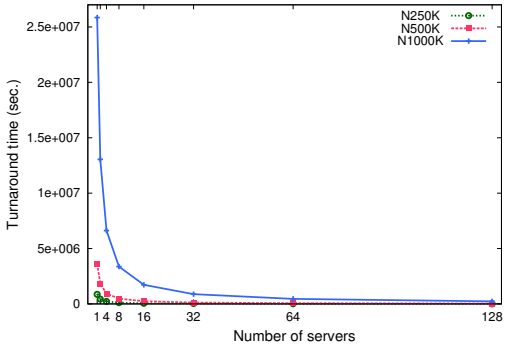
The proposed trajectory mining algorithm has been also tested on synthetic data, in order to carry out a scalability performance analysis on big datasets, up to millions of trajectories traced by hundreds of timestamp points (both high-cardinality and high-dimensional trajectory data). To do that, we developed an ad-hoc data generator allowing to create trajectory data adhering to some suitable parameters (i.e., number of trajectories and timestamps, noise degree, etc.) tuned by input values. A detailed description of the data generator is reported in Section 5.3.1, while scalability analysis on synthetic data is described in Section 5.3.2.

5.3.1 *Synthetic Data Generator: implementation details.*

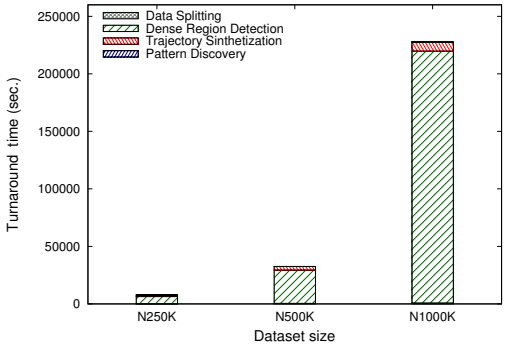
The goal of the data generator is to build trajectory data set, whose characteristics (number of trajectories, patterns, noise degree, etc.) can be tuned by input values. In particular, the trajectory generation process has been made parametric to the following parameters: the number N and the average length L of trajectories, the number R of base regions for each timestamp, the number P of patterns to force within the data, the percentage θ of outlier regions and the degree ρ of overlapping among regions of different patterns. The θ value represents the percentage of regions (with respect to their total number) that will be considered as noise and will populate the outlier set, while ρ is the probability that noise regions will be used during the trajectory generation step.

The synthetic trajectory data generation process works as follows. First, a set of pre-defined dense regions, $\mathcal{BRS}$, and a set of outlier regions, $\mathcal{ORS}$ are generated. The elements in the first group, named also *target* or *oracle regions*, are considered as ground truth areas and will be exploited as base regions for the synthetic mobility patterns, while the elements in $\mathcal{ORS}$ will be exploited for adding noise and perturbations inside data. More formally, the goal is generating trajectories over the timestamp sequence $T =< t_1, t_2, \ldots, t_H >$, where $t_h < t_{h+1}, \forall_{0<h<H}$. Initially, two sets of regions are generated, the *base region set* $\mathcal{BRS} = \{BR_{t_1}, \ldots, BR_{t_H}\}$ and the *outlier region set* $\mathcal{ORS} = \{OR_{t_1}, \ldots, OR_{t_H}\}$, where each BR_{t_i} and OR_{t_i} represents the (base and outlier, respectively) region sets for the timestamp t_i. Base regions are distinctive for the patterns, while outlier regions will be used to create noise and perturbations during the trajectory generation. Set sizes are fixed such that outlier regions amount to almost θ percent of the total number of regions: i.e., $|OR_{t_i}| = \theta \cdot (|OR_{t_i}| + |BR_{t_i}|)$, for each timestamp t_i. To assign at each pattern a set of regions that can be distinctive for it, all the base regions are equally partitioned among patterns, i.e. $BR_{t_i} = \{BR^1_{t_i}, \ldots, BR^P_{t_i}\}$, for each timestamp t_i. A base region $BR^p_{t_i}$ represents the region (exclusively) belonging to the pattern p at time t_i. For such a reason, there is no spatial intersection among regions referring to the same timestamp, i.e. $BR^{p_k}_{t_i} \cap BR^{p_j}_{t_i} = \emptyset$, for each pattern $p_k \neq p_j$ and $BR^{p_k}_{t_i} \cap OR_{t_i} = \emptyset, \forall_{1 \leq k \leq P}$.

After such initialization phase, the generation of trajectories proceeds as follows. In order to create a balanced number of trajectories among patterns, for each p^{th} pattern N/P trajectories are created, where the length L of each trajectory is fixed from a normal distribution with mean H and fixed variance. A trajectory belonging to the pattern p is generated



(a) Turnaround time vs the number of available servers



(b) Turnaround time (with the partial times required by each step), on 128 servers

Fig. 13. Synthetic Data Results. Turnaround time for different data sizes.

as follows: for each timestamp t_i, with $i = 1, \ldots, L \leq H$, ρ percent of points are randomly picked from the outlier region set OR_{t_i}, while the remaining from the base region set $BR^p_{t_i}$. The final result is a dataset composed of N trajectories, built from P generative patterns, with ρ degree of overlapping (i.e., noise) among regions of different patterns. Table 1 reports the input parameters for the data generator, with a brief description and the range of values fixed during the experiments.

TABLE 1
Trajectory Generation Parameters

Parameter	Meaning	Values
N	No. of trajectories to be generated	250K-1M
L	Average length of trajectories	16-128
R	No. of base regions	10
P	No. of generative trajectory patterns	10
ρ	Noise degree (%)	0-15
θ	Overlap degree (%)	5-20

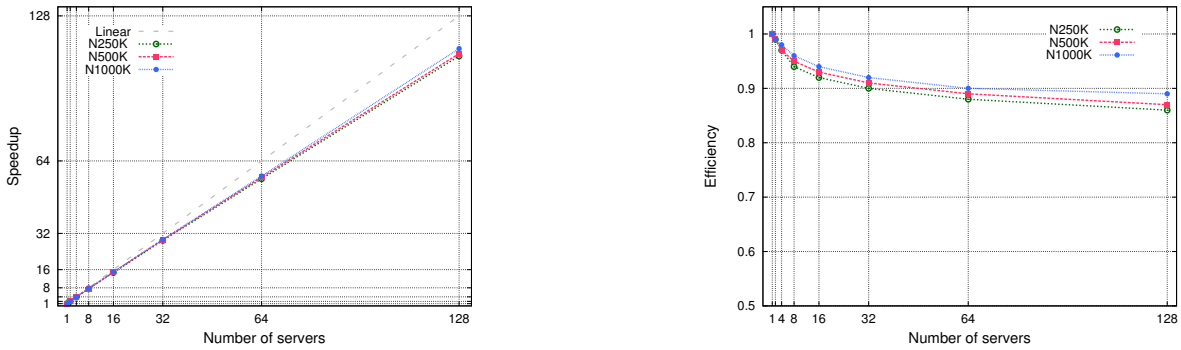
5.3.2 *Experimental Results.*

We generated several trajectory data sets through our ad-hoc generator. In particular, since we can increase the problem size in either the number of data points N and the number of timestamps (or dimensions) T, we can study the scalability with respect to both parameters. In addition, it is interesting to analyze how the number of outlier regions as well as the noise introduced in the data can affect the algorithm results. We report three sets of experiments, where we vary N, T and the couple (ρ,θ) values, respectively.

Varying *N*. First, we study scalability and speed-up performances when the number of trajectories is varied. We generated three different datasets by varying N from 250K to 1000K. All the other parameters have been fixed as follows: $T = 128$, $R = 10$, $\theta = 20\%$, $\rho = 10\%$, $P = 10$. We tag such datasets as $N250K$, $N500K$ and $N1000K$. After some preliminary experiments carried on the Cloud, we have realized that it is not feasible to analyze such high cardinality datasets on a small number of nodes. For example, the execution time of a single instance of DBScan to analyze a one-timestamp partition of $N500K$ dataset takes on average 8 hours. So, the sequential time to analyze 128 partitions would take around 41 days. For such a reason, we carried out real experiments with 32, 64 and 128 Cloud nodes and analytically inferred the time needed to perform the whole analysis by exploiting a lower number of nodes.

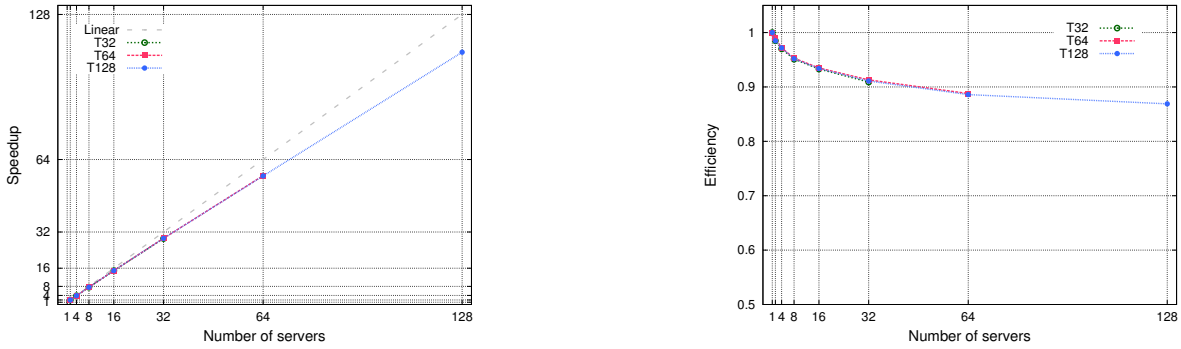
Figure 13(a) shows the turnaround times of the whole discovery pattern process for the three considered cases, using from 1 up to 128 virtual servers. We can observe that the turnaround time strongly decreases for all datasets with the number of available servers. In particular, it is impressive that the time needed to analyze one million trajectories (i.e., $N1000K$) would require 7181 hours (about 299 days) of processing time on a single server, while it decreases to only 63 hours (less than three days) on 128 servers. Figure 13(b) shows the parallel execution time varying the number N of trajectories, achieved by exploiting 128 virtual servers. In particular, it is shown the time required by each step of the analysis process. It is evident that the dense region detection takes more than 99% of the total execution time, with a quadratic order increasing with respect to the data sizes N (according to the temporal complexity of the DBScan algorithm). The synthesization step has a linear increasing trend with respect to N, but its contribution to the total time amount to less than 1%. The other two steps, i.e. data splitting and pattern discovery, take a negligible time.

Figure 14(a) shows the execution speed-up values, for different number of trajectories. The speed-up is almost linear with all datasets, up to the case of 32 nodes. Anyhow, for a higher number of nodes, it maintains a notable trend. On the $N1000K$ dataset it achieves the value of 113.54 for 128 nodes. This result shows a high scalability of both the framework and the parallelization strategy. Figure 14(b) shows the application efficiency, vs the number of servers and for different number of trajectories. As shown in the figure, efficiency maintains a good trend and notable values even for high number of running servers. For example, for the largest dataset the efficiency on 32 servers is equal to 0.92 whereas on 128 servers it is equal to 0.89. So, the 92% and 89% of the computing power of each used server is exploited, respectively. Moreover, it is noticed that the efficiency increases with the trajectory length (fixed the number of servers). This means that the distributed architecture is increasingly convenient when the problem size increases,



(a) Speed-up (b) Efficiency

Fig. 14. Synthetic Data Results. Speed-up and efficiency vs the number of available servers, for different data sizes.



(a) Speed-up (b) Efficiency

Fig. 15. Synthetic Data Results. Speed-up and efficiency vs the number of available servers, for different number of timestamps (Dataset N500K).

which is a another sign of good scalability properties.

Varying *T*. Second, we study the speed-up behavior when the number of timestamps T tracing the trajectories is varied. We generated four different trajectory datasets by varying the number T of timestamps from 16 to 128. All the other parameters have been fixed as follows: $N = 500K$, $R = 10$, $\theta = 20\%$, $\rho = 10\%$, $P = 10$. We label such datasets as $T16$, $T32$, $T64$ and $T128$. Figure 15(a) shows the execution speed-up values. The speed-up is linear with all datasets, up to the case of 32 nodes. In particular, for the 32 timestamp dataset, the speed-up increases from 1.97 using 2 servers to 29.10 using 32 servers, that represents a very notable trend. It is worth noting that the speed-up keeps a good trend even for higher data dimensionality. In fact, for the 128 timestamp case, it increases from 1.97 using 2 servers to 111.33 exploiting 128 servers. This results in a very good scalability of the framework, even with respect to trajectories traced by a high number of timestamps. Figure 15(b) shows the efficiency, vs the number of servers and for different trajectory lengths (i.e., number of timestamps). It shows notable values, both for high number of running servers and high data dimensionality. For example, for the 128 timestamps case ($T128$), the achieved efficiency on 32, 64 and 128 servers results equal to 0.91, 0.88 and 0.86, respectively. These values show good scalability and efficiency of the framework, even when processing high-dimensional data.

Varying ρ and θ. Finally, let us study the results when the overlap (ρ) and outlier (θ) degrees are varied. As described in Section 5.3.1, θ is the percentage of regions in R that will be considered as noise (and that are assigned to the outlier set), while ρ is the probability that points belonging to noise regions are added to the trajectories (during the generation step). First, we generated several datasets by varying ρ, from 0% to 20%. Figure 16 shows the execution time of Dense Regions Detection step on such datasets, versus ρ and for different data set sizes. It is worth noting that the elapsed time does not have notable changes with respect to ρ variations, nevertheless it is obviously strongly dependent on the number of trajectories to be analyzed.

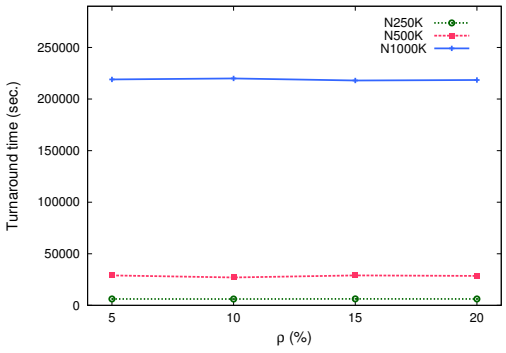
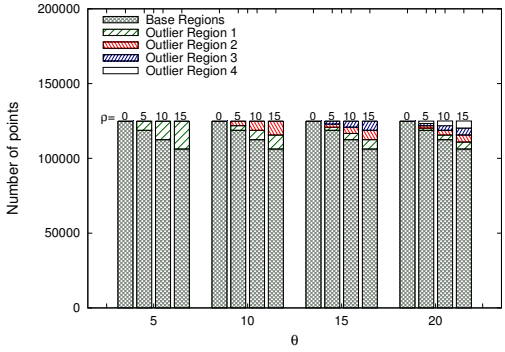
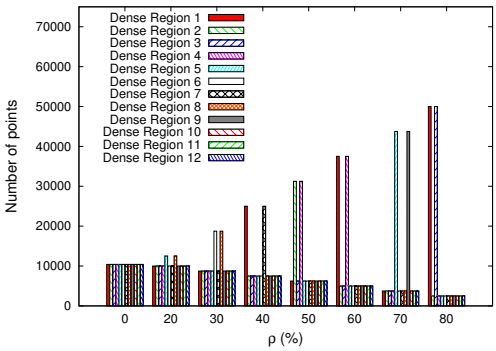


Fig. 16. Synthetic Data Results. Turnaround time of the Dense Region Detection step vs ρ, for different data sizes



(a) Distribution of points among base regions and outlier regions, varying ρ and θ



(b) Dense Region sizes vs ρ, for $\theta = 20\%$

Fig. 17. Synthetic Data Results. Distribution of points among dense regions and outlier regions, varying ρ and θ (N=125K, R=12).

Now, we analyze the results when varying both ρ and θ. To do that, we generated several datasets, with ρ varying from 0% to 80% and θ from 5% to 20%. All the other parameters have been fixed as follows: $N = 125K, T = 10$, $R = 12$, $P = 10$. We run the algorithm on such datasets. Figures 17(a) and 17(b) show the distribution of points among dense regions, after the dense regions detection step.

In particular, each histogram in Figure 17(a) shows the number of points that have been assigned to base regions and outlier regions, with different values of ρ and θ. The main conclusion is that the dense region detection step is able to discover dense regions adhering to the generative patterns; in particular, the main issues that can be derived from these plots are:

- considering the particular no-noise case, i.e. $\rho = 0$, all the points are assigned to base regions; in this case, in fact, independently from the number of outliers (θ), data are generated by selecting the points only from the base region set; during the processing step, the density-based clustering algorithm is able to discover the generative dense regions in an accurate manner;
- the noise degree ρ influences the percentage of points that are noisy with respect to the total number, while θ leads the number of regions noise is distributed on; this is clearly highlighted by the histograms corresponding to $\theta = 5\%, 10\%$ and 15%: it is evident that for a fixed ρ, higher is θ, greater is the number of discovered regions hosting noisy points.

On the other side, Figure 17(b) shows the number of points that have been assigned to the 12 discovered regions, varying ρ and for $\theta = 20\%$. The figure shows that in the no-noise case ($\rho = 0$) the clusters are quite balanced in the number of grouped points. For higher values of ρ the produced clusters are more and more unbalanced in size, resulting in few big clusters (grouping most of the points of the considered trajectories) and several smaller non significant ones (that actually do no represent dense regions). As such, from those smaller clusters it is not possible to extract trajectory patterns.

6 Conclusion

In the paper we presented a Cloud-based software environment specifically designed for urban computing supporting smart city applications. The environment has been designed as a composition of different services allowing users to gather and collect environmental data, and process and analyze them in order to mine social and urban behaviors. Within such system we have designed and implemented a parallel methodology, modeled by the workflow formalism, for pattern discovery from trajectory data. The main idea of the methodology consists in *(i)* finding the more densely passed through regions in a given geographical area, and *(ii)* then extracting trajectory patterns from those regions in the form of association rules. In particular, the applicative scenario described in the paper focuses on the study of trajectories followed by vehicles in an urban area with the aim to discover knowledge models, and thus to catch users'mobility behaviors. The paper describes in details the design of the workflow implementing the application and its execution by a workflow engine integrated in the environment. Experimental evaluation of the framework, performed both on a real-world dataset (concerning mobility of citizens within an urban area) and on several synthetic datasets, shows the scalability and efficiency of the approach. Specifically, the trajectory pattern mining process takes advantage from a Cloud architecture in terms of both execution time, speed-up and scale-up. It is worth noting that the Cloud system allows developers and users to analyze large amount of trajectory data whose size is much higher than that can be analyzed by the most of the systems found in literature.

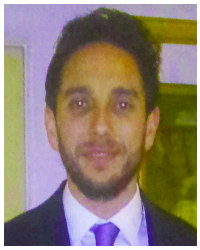
As future work, we plan to extend some functionalities of the system for implementing new urban computing applications, including the following ones: *(i) predict the future location* of a moving vehicle based on the trajectory pattern models obtained from the analysis; *(ii) travel recommendations* exploiting mobility models to suggest the best routes that vehicles can follow to reach a given place; *(iii) intelligent traffic management* predicting traffic congestion patterns to reduce the wasted time due to vehicular mobility. Finally, the trajectory mining algorithm will be modified and extended to include automatic parameter setting in the dense region discovery step of the mining workflow.

Acknowledgments

This research work has been partially funded by the MIUR project DICET-INMOTO (PON04a2_D).

REFERENCES

[1] Y. Zheng, L. Capra, O. Wolfson, and H. Yang, "Urban computing: Concepts, methodologies, and applications," *ACM Transactions on Intelligent System and Technologies*, vol. 5, no. 3, pp. 38:1–38:55, 2014.
[2] "ESS." [Online]. Available: www.ess.co.at/
[3] "EVO." [Online]. Available: www.evo-uk.org/
[4] "Ibm smarter city solutions on cloud." [Online]. Available: www-01.ibm.com/software/industry/smartercities-on-cloud/
[5] E. Cesario, C. Comito, and D. Talia, "Towards a cloud-based framework for urban computing. the trajectory analysis case." in *Proc. of the 3rd Int. Conference on Cloud and Green Computing*, ser. CGC '13. IEEE, 2013, pp. 16–23.
[6] J. Lee, J. Han, and X. Li, "A unifying framework of mining trajectory patterns of various temporal tightness," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 6, pp. 1478–1490, 2015.
[7] Y. Zheng, "Trajectory data mining: An overview," *ACM Trans. Intell. Syst. Technol.*, vol. 6, no. 3, pp. 1–41, 2015.
[8] A. Altomare, E. Cesario, C. Comito, F. Marozzo, and D. Talia, "Using clouds for smart city applications," in *Proc. of the fifth Int. Conference on Cloud Computing Technology and Science*, ser. Cloudcom'13. IEEE, 2013, pp. 234–237.
[9] A. Altomare, E. Cesario, C. Comito, F. Marozzo, and D. Talia, "Trajectory pattern mining over a cloud-based framework for urban computing," in *Proc. of the 2014 IEEE Int. Conference on High Performance Computing and Communications*, 2014, pp. 367–374.
[10] "EPIC." [Online]. Available: www.epic-cities.eu/
[11] "Life2.0." [Online]. Available: www.life2project.eu
[12] N. Mamoulis, H. Cao, G. Kollios, M. Hadjieleftheriou, Y. Tao, and D. W. Cheung, "Mining, indexing, and querying historical spatiotemporal data," in *Proc. of the tenth ACM SIGKDD Int. Conference on Knowledge Discovery and Data Mining*, ser. KDD '04. ACM, 2004, pp. 236–245.
[13] H. Jeung, Q. Liu, H. Shen, and X. Tao Zhou, "A hybrid prediction model for moving objects," in *Proc. of the 2008 IEEE 24th Int. Conference on Data Engineering*, ser. ICDE '08. IEEE Computer Society, 2008, pp. 70–79.
[14] J. Yuan, Y. Zheng, X. Xie, and G. Sun, "T-drive: Enhancing driving directions with taxi drivers' intelligence," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 1, pp. 220–232, 2013.
[15] F. Giannotti, M. Nanni, F. Pinelli, and D. Pedreschi, "Trajectory pattern mining," in *Proc. of the 13th ACM SIGKDD Int. Conference on Knowledge Discovery and Data Mining*, ser. KDD '07. ACM, 2007, pp. 330–339.
[16] F. Marozzo, D. Talia, and P. Trunfio, "A workflow-oriented language for scalable data analytics," in *Proc. of the First Int. Workshop on Sustainable Ultrascale Computing Systems (NESUS 2014)*, Porto, Portugal, 2014.
[17] E. Cesario, M. Lackovic, D. Talia, and P. Trunfio, "Programming knowledge discovery workflows in service-oriented distributed systems," *Concurrency and Computation: Practice and Experience*, vol. 25, no. 10, pp. 1482–1504, 2013.
[18] F. Marozzo, D. Talia, and P. Trunfio, "Js4cloud: Script-based workflow programming for scalable data analysis on cloud platforms," *Concurrency and Computation: Practice and Experience*, 2015.
[19] F. Marozzo, D. Talia, and P. Trunfio, "A cloud framework for parameter sweeping data mining applications," in *Proc. of the 3rd IEEE Int. Conference on Cloud Computing Technology and Science (CloudCom 2011)*, 2011, pp. 367–374.
[20] M. Ester, H. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proc. of the second Int. Conference on Knowledge Discovery and Data Mining*, ser. KDD'96. AAAI Press, 1996, pp. 226–231.
[21] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules in large databases," in *Proc. of the 20th Int. Conference on Very Large Data Bases*, ser. VLDB '94. Morgan Kaufmann Publishers Inc., 1994, pp. 487–499.
[22] J. Yuan, Y. Zheng, C. Zhang, W. Xie, X. Xie, G. Sun, and Y. Huang, "T-drive: driving directions based on taxi trajectories," in *Proc. of the 18th SIGSPATIAL Int. Conference on Advances in Geographic Information Systems*, ser. GIS '10. ACM, 2010, pp. 99–108.
[23] J. Yuan, Y. Zheng, X. Xie, and G. Sun, "Driving with knowledge from the physical world," in *Proc. of the 17th ACM SIGKDD Int. Conference on Knowledge Discovery and Data Mining*, ser. KDD '11. ACM, 2011, pp. 316–324.



Albino Altomare received a Master degree in computer engineering from the University of Calabria, Italy, in 2012. Since 2012 he is serving as research fellow at the Institute of High Performance Computing and Networks (ICAR-CNR) of the National Research Council (CNR) of Italy. ICAR-CNR. He coauthored several papers published in conference proceedings. His research interests include Knowledge Discovery applications and energy-aware Cloud Computing.

Eugenio Cesario is a researcher at ICAR-CNR, the Institute of High Performance Computing and Networks of the National Research Council of Italy. He co-authored more than 40 scientific papers in international journals, conference proceedings, and edited volumes. He served as a chair, organizer, or program committee member of several international conferences. His research interests include Distributed Data Mining, Cloud computing, Knowledge Discovery applications and Energy-aware architectures.

Carmela Comito is a researcher at the Institute of High Performance Computing and Networking of the Italian National Research Council (ICAR-CNR), Italy. In 2006 she was a visiting researcher at the School of Computer Science of the University of Manchester, UK. She co-authored several scientific papers in international journals, conference proceedings, and edited volumes. She served as a chair, program committee member and reviewer of several international conferences. Her current research interests include mobile and ubiquitous computing, social network data analysis and mining.

Fabrizio Marozzo received a Master degree in computer engineering and a Ph.D. in systems and computer engineering from the University of Calabria, Italy, in 2009 and 2013, respectively. In 2011-2012 he visited the Barcelona SuperComputing Center (BSC) for a research internship. He coauthored several papers published in conference proceedings, edited volumes and international journals. He has been a member of the program committee of scientific conferences and edited volumes. His research interests include distributed systems, Cloud computing, and peer-to-peer networks.

Domenico Talia is a professor of computer engineering at the University of Calabria. His research interests include parallel and distributed data mining algorithms, Cloud computing, Grid services, distributed knowledge discovery, peer-to-peer systems, and parallel programming models. He is a member of the editorial boards of IEEE Transactions on Cloud Computing, Future Generation Computer Systems, the International Journal of Web and Grid Services, the Journal of Cloud Computing - Advances, Systems and Applications, Scalable Computing: Practice and Experience, the International Journal of Next-Generation Computing.