Trajectory Pattern Mining over a Cloud-based Framework for Urban Computing

Albino Altomare*, Eugenio Cesario*, Carmela Comito*†, Fabrizio Marozzo† and Domenico Talia*†
*ICAR-CNR, {altomare,cesario}@icar.cnr.it
†DIMES-University of Calabria, {ccomito,fmarozzo,talia}@dimes.unical.it

***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 information can be analyzed to extract descriptive and predictive models that can be properly exploited to improve urban life. This paper presents a workflow-based parallel approach for discovering patterns and rules from trajectory data, executed on a Cloud-based framework for urban computing. Experimental evaluation shows that, due to complexity and large data involved in the application scenario, the trajectory pattern mining process takes advantage from the scalable execution environment offered by a Cloud architecture.**

I. Introduction

Urban computing is the process of acquisition, integration, and analysis of big and heterogeneous urban data to tackle the major issues that cities face, 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. In addition, managing heterogeneous data volumes while allowing interoperability among different tools, it also needs compliance to standards. In this regard, Cloud computing systems are suitable platforms to fulfil most of the above requirements, due to their features such as scalable computing, on-demand processing, facilitating data accessibility and storage across platforms [1], [2], [3], [4].

Accordingly to this aim, we developed a Cloud-based framework specifically designed for solving urban computing issues in 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 and collect environmental data, process and analyze them in order to mine social and environmental behaviors.

The discovery of mobility models is one of the most challenging issues in urban computing. It can improve resource furnishing and management of cities. In addition, the increasing pervasiveness of mobile devices along with the use of wireless and GPS sensors is making the monitoring of people and vehicle movements a very common task. This leads to the generation of a large number of trajectories drawn by the users during their daily activities. Such amount of information can be analyzed to discover people and community behavior, i.e. patterns, rules and regularities in moving trajectories. The basic assumption is that people often tend to follow common routes: e.g., they go to work every day travelling the same roads. Thus, if we have enough data to model typical behaviors, such knowledge can be used to predict and manage future movements of people [6], [7]. For example, data generated by using mobile devices produce patterns of movement that can 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 [5]. In particular, the applicative scenario described here focuses on the study of the trajectories followed by mobile devices 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. We apply the trajectory pattern extraction methodology to a real-world dataset concerning mobility of citizens within an urban area. Experimental evaluation, carried out on a public Cloud platform, shows that, due to complexity and large data involved in the application scenario, the trajectory pattern mining process takes advantage from the scalable execution platform offered by the Cloud.

For the sake of clarity, this paper extends a research work whose some activities have been introduced in [4] and in [5]. In particular, [5] presents the Cloud-based architecture for Urban Computing, while paper [4] (a poster paper) gives general hints on the exploitation of Clouds for smart city applications. Differently from the previous ones, this paper is focused on the development and execution of the trajectory mining methodology on a public Cloud [5], and it provides several original contributions with respect to the previous ones. First, it describes in details the design of the workflow

implementing the application and its execution by a workflow engine. Second, it shows the results of a complete experimental evaluation on a public Cloud, by pointing out advantages in terms of execution time, speedup and scaleup.

The rest of the paper is organized as follows. Section II outlines related work in the area of mobility pattern discovery, with a particular focus on Cloud-based urban computing. Section III reports a short description of the Cloud-based framework. Section IV presents the trajectory analysis scenario describing the trajectory pattern detection methodology together with the designed workflow. Experimental evaluation on a real use case scenario is reported in Section V. Section VI concludes the paper.

II. Related Work

The objective of this work is twofold: (i) provide a Cloud-based framework for efficiently manage socio-environmental data, with a particular focus on the urban context of cities, and (ii) provide 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 and projects in both the areas.

A. 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) [1] 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) [2] uses Clouds to achieve similar objectives in the soil and water domains.

The European Platform for Intelligent Cities (EPIC) [8] combines the 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 [9] 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 between 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 [3]. 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.

B. Trajectory Pattern Mining

Discovering periodic patterns from historical object movements is a very challenging task. In [7] 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 [6]. 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 [10] 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 [11] the authors extend the sequential pattern mining methodology to analyse 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, this work is novel in two aspects. First, it is a pioneering workflow-based approach to mine trajectory patterns on a real public Cloud platform. This allows 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. Second, the experimental tests conducted on these large datasets show that the whole process takes advantages from such a scalable environment, both in terms of execution time and achieved speedup. Moreover, another important distinguishing feature of our framework is that it has been tailored to provide general-purpose services for urban planning and management within the city context. Nevertheless, the framework has been designed as a set of modular components allowing easy extensibility and integration of different heterogeneous components (e.g., software, data sources, etc).

III. A Cloud-based framework for Urban Computing

In this section we introduce the architecture of the framework 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 substrate allowing for 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 to implement the above urban-related services: facilitates data access and storage across platforms, provides on-demand computational resources, and allows 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 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 design 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].

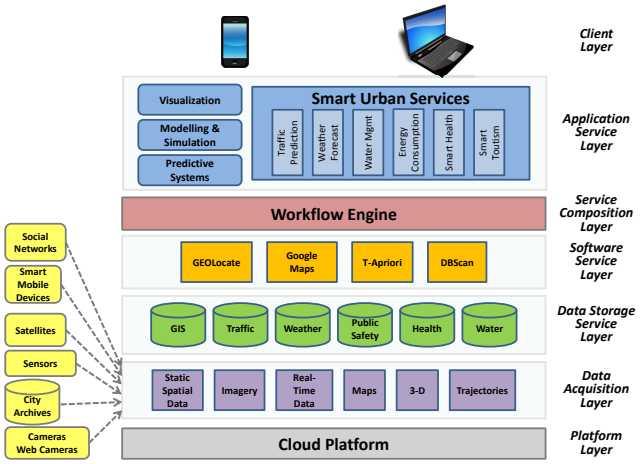


Fig. 1. A Cloud-based architecture for urban computing.

The implementation of the Service Composition Layer has been done using the Data Mining Cloud Framework [12], a software environment that allow users to design and execute data analysis, mining and knowledge discovery workflows on the Cloud. Following the approach proposed in [13], such a framework models knowledge discovery workflows as graphs whose nodes represent resources (datasets, data mining tools, data mining models) and whose edges represent dependencies between resources. The framework includes a Website to compose workflows and to submit their execution to the Cloud, following a Software-as-a-Service approach.

Figure 2 shows the architecture of the Data Mining Cloud Framework, which includes a set of binary and text data containers used to store data to be mined (*Input datasets*) and the results of data mining tasks (*Data mining models*), a *Task Queue* that contains the workflow tasks to be executed, a *Task Table* and a *Tool Table* that keep information about current tasks and available tools, a pool of k *Workers* (k is the number of virtual servers available) in charge of executing the workflow tasks and finally a *Website* that allows users to submit, monitor the execution, and access the results of knowledge discovery workflows.

The following steps are performed to develop and execute a knowledge discovery application [14] (see Figure 2):

1) A user accesses the Website and develops her/his application as a workflow through an HTML-5 interface.
2) After application submission, a set of tasks that compose the workflow are created and inserted into the Task Queue.
3) Each idle Worker picks a task from the Task Queue, and starts its execution on a virtual server.
4) Each Worker gets the input dataset from its original location.
5) After task completion, each Worker puts the result on a data storage element.
6) The Website notifies the user as soon as her/his task(s) have completed, and allows her/him to access the results.

The Data Mining Cloud Framework has been designed to be implemented on different Cloud systems. The current implementation is based on Windows Azure ("www.microsoft.com/windowsazure").

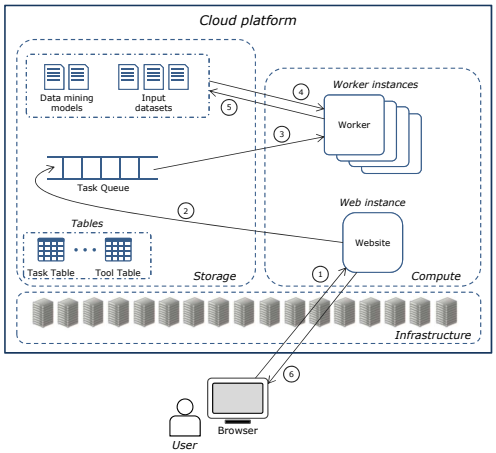


Fig. 2. Architecture of the Data Mining Cloud Framework.

IV. 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 mobile devices, 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 various ways. The trajectory patterns extracted represent a basic building block around which further tasks can be implemented, including the following ones:

- *Next location prediction.* Predict the possible future location of a moving object, based on the object recent movements and trajectory pattern models, to anticipate or pre-fetch possible services in that location.
- *Intelligent traffic management.* Predict traffic congestion patterns and adopt improvements to 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 their 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. Second, 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).

A. 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. $R_{t_1}^{j_1} \wedge R_{t_2}^{j_2} \wedge \ldots \wedge R_{t_r}^{j_r}$ is the primes, while $R_{t_s}^{j_s}$ 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, that 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.

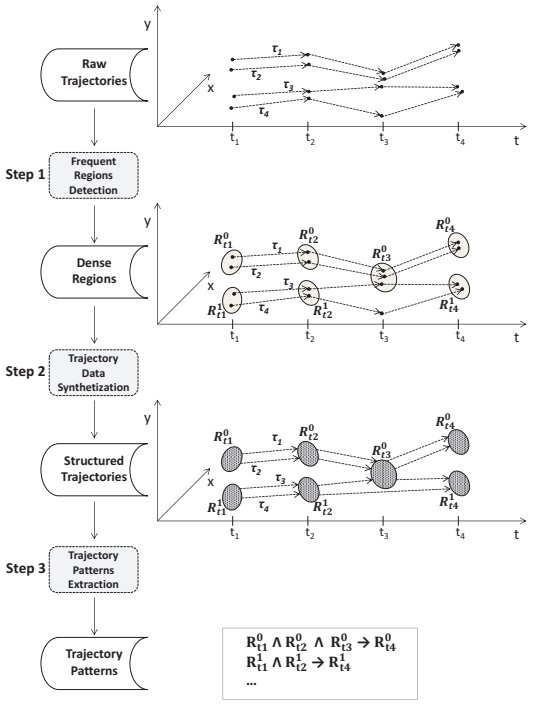


Fig. 3. Trajectory Pattern Detection Steps.

B. Trajectory Pattern Detection: a parallel implementation

The trajectory pattern detection process consists of a sequence of concatenated 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 step is the most time-consuming and critical task. For such a reason, our first 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

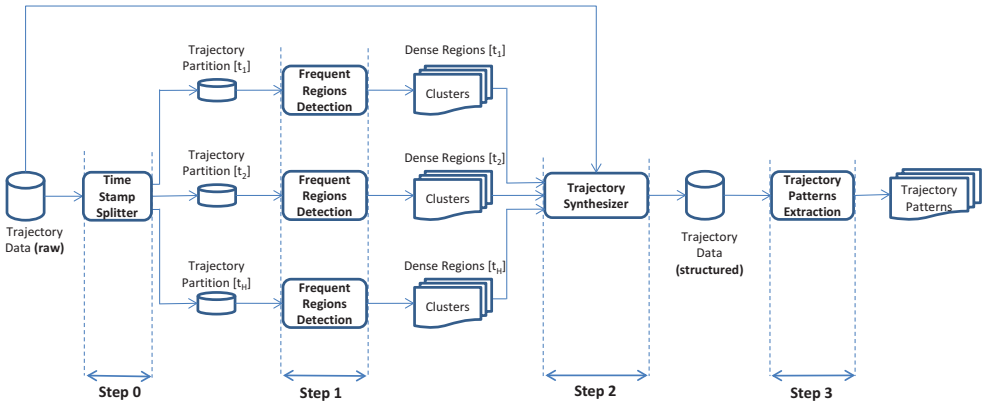


Fig. 4. Trajectory Pattern Detection Workflow.

dependencies among nodes. The original data set D is a raw trajectory data, populated by the trajectories (represented in the previously described format) of some users 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 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. At the end of this step, $|T|$ different datasets are available. It is worth noticing that this is an additional step with respect to the sequential case, where no splitting step is contemplated.

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.

Step 2 - Trajectory Data Synthetization. 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 figure, results 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.

V. Experimental Evaluation

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. The workflow has been composed and executed on the Cloud system described in Section III, exploiting the Data Mining Cloud Framework [12]. 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.

The input dataset chosen for the experiments is the *T-Drive Trajectory Data Sample* [15], [16], 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. The total number of points is about 15 millions 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 D16, D32, D64 and D128, respectively. Those four datasets have been used in the experimental evaluation. For what concerns the algorithms, the *Frequent Regions Detection* step has been implemented by using DBSCAN [17], a density-based clustering algorithm, whereas the *Trajectory Pattern Extraction* step has been performed by T-Apriori, an our ad-hoc modified version of the well-known Apriori algorithm [18].

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. Moreover, some nodes

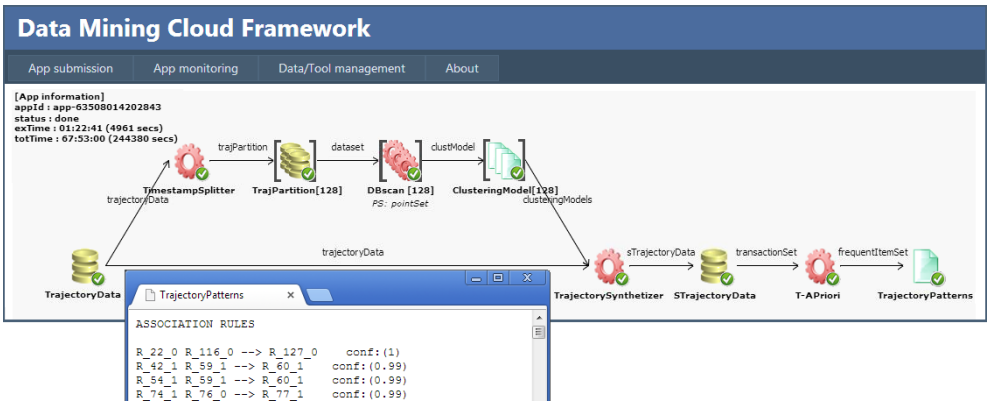


Fig. 5. Trajectories workflow at the end of the execution, with visualization of the final result.

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 reproduces and implements the 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 the 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).

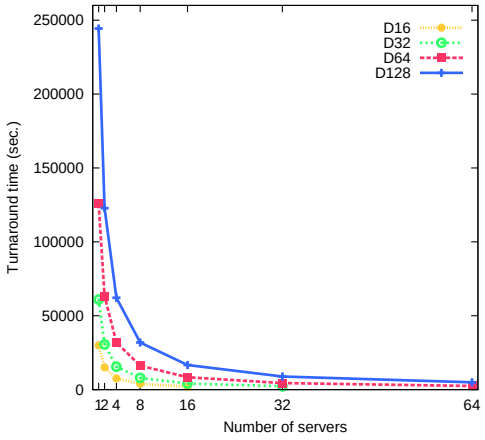
We executed our experiments on the Microsoft Azure platform using 1 virtual server to run the Data Mining Cloud Framework Website, and up to 64 virtual servers for the Workers. Each virtual server was equipped with a single-core 1.66 GHz CPU, 1.75 GB of memory, and 225 GB of disk space. Each test has been executed on the four datasets previously described (i.e., trajectories of different length) and by varying the number of virtual servers used to run the trajectory pattern mining application. As performance indicators, we used the *turnaround time* and the achieved *speedup*. The first one is the total execution time of the distributed algorithm, that is, the elapsed time from the task submission until the final result is returned to it. The second one is the ratio of the turnaround time elapsed by exploiting 1 node to the turnaround time on n nodes.

Figure 6(a) shows the turnaround times of the application for the four considered datasets using from 1 to 64 virtual servers. The case of 1 server corresponds to a sequential architecture in which the mining computation is performed on a single node. Therefore the shown results can also be seen as a comparison between a parallel and a sequential solution. In particular, 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 diminuishes from 17 hours to 38 minutes. For the 64 timestamp dataset the turnaround time decreases from 35 hours to 41 minutes. Finally, with the 128 timestamp, the turnaround time ranges from about 68 hours to about 1.4 hour using 64 virtual machines.

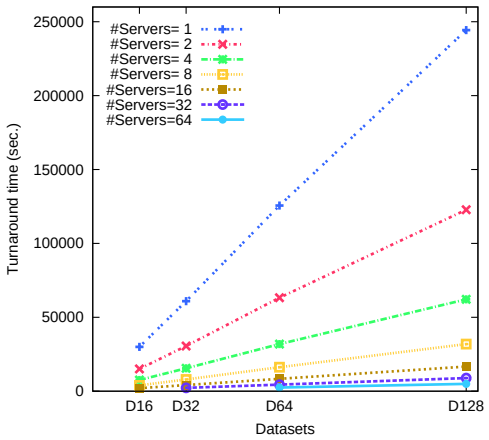
Figure 6(b) shows how the turnaround time increases with respect to the dataset size, for a different number of virtual machines. The graph shows that the time required to execute the entire workflow increases proportionally with the increase of the input size. On the contrary, the time required to execute the entire workflow decreases proportionally with the increase of computing resources.

Figure 7(a) shows the execution speedup values. The speedup is almost linear with all datasets, up to the case of 16 nodes. In particular, for the 16 timestamp dataset, the speedup passes from 2.0 using 2 servers to 14.5 using 16 servers, that represents a very notable trend. For an higher number of nodes, the speedup is not linear because of the influence of the sequential steps of the application, however it follows a good trend. In fact, for the 32 timestamp dataset, it ranges from 2.0 to 26.9 using 32 servers, while for the 64 timestamp dataset the speedup ranges from 2.0 to 50.7. Finally, with the 128 timestamp it ranges from 2.0 to 49.3 by using 64 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 timestamp on 16 servers, 32 timestamp on 32 server servers, 64 timestamp on 64 servers). The results show that the total turnaround time is almost constant. 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

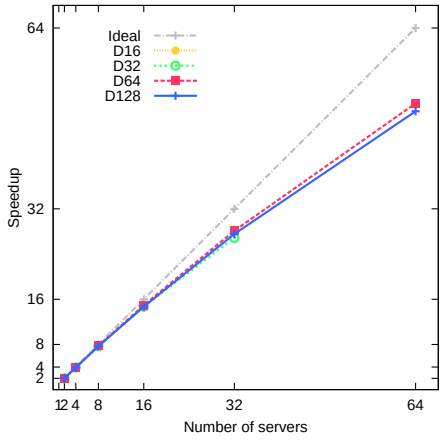


(a) Turnaround time vs the number of available servers, for different data sizes.

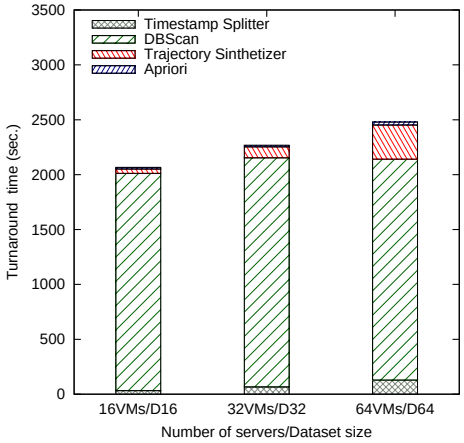


(b) Turnaround times vs data sizes, for different number of available servers.

Fig. 6. Turnaround times in different scenarios.



(a) Speedup vs the number of available servers, for different dataset sizes.



(b) Scale-up with the partial times required by each step of the workflow.

Fig. 7. Speedup and Scaleup.

can notice that in each scenario the DBSCAN step takes most of the total time and this time is almost constant in all three scenarios. This is due to the fact that the parallelization degree of DBSCAN executions increases proportionally with the dataset size. On the other side, the time required by Timestamp Splitter, Trajectories Synthetizer and Apriori steps, that are implemented as sequential tasks, increases for larger dimensions of the dataset.

Finally, we evaluate 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 8, 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 2066 seconds, while the 64 timestamp dataset takes 172 seconds on a total of about 2479 seconds. This means that the system overhead is just the 2,3% and the 6,9% of the total execution.

Through the Visualization module of the proposed framework we obtained the visualization of trajectories on the city map plus the taxi movement rules like the snapshot in Figure 9, which shows a graphical visualization of a trajectory pattern rule discovered during the trajectory data analysis. In particular, is illustrated the rule $R_1^{23} \wedge R_4^{11} \wedge R_6^{13} \xrightarrow{0.75} R_{12}^{28}$, which models a relation between the first three dense regions occurring in the premise of the rule and the dense region occurring in the consequent part. The rule shows that if a taxi passes through the dense regions R^{23}, R^{11}, R^{13} at the

timestamps 1,4 and 6 respectively, it will pass for the region R^{28} at the timestamp 12 with a confidence of 75%.

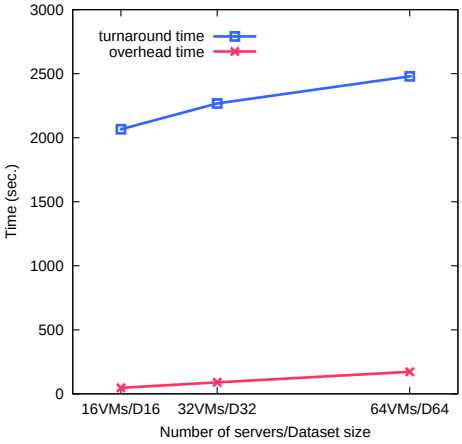


Fig. 8. Overhead time vs data sizes.

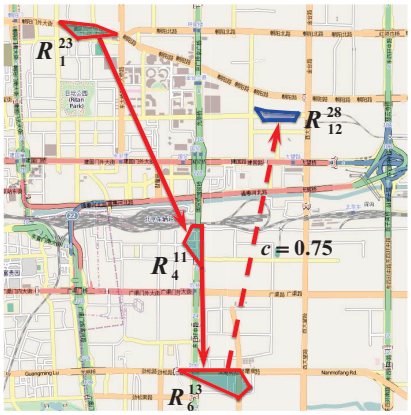


Fig. 9. Graphical visualization of a trajectory pattern rule discovered from Beijing taxi trajectory data.

VI. Conclusion

A large amount of movement data is daily collected, due to the increasing pervasiveness of mobile and wireless devices, sensing technologies, GPS traces, and sensors. Such collection of information can be analyzed to discover descriptive and predictive models, that can be exploited to have a smart management of the city resources. To this aim, we developed a Cloud-based framework specifically designed for urban computing supporting smart cities. The framework has been designed as a composition of different services allowing to gather and collect environmental data, and to process and analyze them in order to mine social and urban behaviors. Within such framework 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.

Experimental evaluation of the framework, conducted on a real-world dataset, shows that the trajectory pattern mining process takes advantage from a Cloud architecture in terms of both execution time and speedup.

As future work, we will extend some functionalities of the framework, and we will implement on it new urban computing applications. In particular, in the next development steps we will introduce some optimizations in the trajectory analysis methodology with the goal of exploiting further sources of parallelism.

References

[1] "ESS." [Online]. Available: www.ess.co.at/

[2] "EVO." [Online]. Available: www.evo-uk.org/

[3] "Ibm smarter city solutions on cloud." [Online]. Available: www-01.ibm.com/software/industry/smartercities-on-cloud/

[4] A. Altomare, E. Cesario, C. Comito, F. Marozzo, and D. Talia, "Using clouds for smart city applications," in *Proceedings of the fifth International Conference on Cloud Computing Technology and Science*, ser. Cloudcom'13. IEEE, 2013, pp. 234–237.

[5] E. Cesario, C. Comito, and D. Talia, "Towards a cloud-based framework for urban computing. the trajectory analysis case." in *Proceedings of the 3rd International Conference on Cloud and Green Computing*, ser. CGC '13. IEEE, 2013, pp. 16–23.

[6] H. Jeung, Q. Liu, H. Shen, and X. Tao Zhou, "A hybrid prediction model for moving objects," in *Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*, ser. ICDE '08. IEEE Computer Society, 2008, pp. 70–79.

[7] N. Mamoulis, H. Cao, G. Kollios, M. Hadjieleftheriou, Y. Tao, and D. W. Cheung, "Mining, indexing, and querying historical spatiotemporal data," in *Proceedings of the tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '04. ACM, 2004, pp. 236–245.

[8] "EPIC." [Online]. Available: www.epic-cities.eu/

[9] "Life2.0." [Online]. Available: www.life2project.eu

[10] 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.

[11] F. Giannotti, M. Nanni, F. Pinelli, and D. Pedreschi, "Trajectory pattern mining," in *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '07. ACM, 2007, pp. 330–339.

[12] F. Marozzo, D. Talia, and P. Trunfio, "Using clouds for scalable knowledge discovery applications," in *Euro-Par Workshops*. Springer, 2012, pp. 220–227.

[13] 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.

[14] F. Marozzo, D. Talia, and P. Trunfio, "A cloud framework for parameter sweeping data mining applications," in *Proceedings of the third International Conference on Cloud Computing Technology and Science*, ser. CloudCom'11. IEEE, 2011, pp. 367–374.

[15] J. Yuan, Y. Zheng, C. Zhang, W. Xie, X. Xie, G. Sun, and Y. Huang, "T-drive: driving directions based on taxi trajectories," in *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*, ser. GIS '10. ACM, 2010, pp. 99–108.

[16] J. Yuan, Y. Zheng, X. Xie, and G. Sun, "Driving with knowledge from the physical world," in *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '11. ACM, 2011, pp. 316–324.

[17] M. Ester, H. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the second International Conference on Knowledge Discovery and Data Mining*, ser. KDD'96. AAAI Press, 1996, pp. 226–231.

[18] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules in large databases," in *Proceedings of the 20th International Conference on Very Large Data Bases*, ser. VLDB '94. Morgan Kaufmann Publishers Inc., 1994, pp. 487–499.