Distributed computation of mobility patterns in a smart city environment

Eugenio Cesario, Franco Cicirelli, Carlo Mastroianni

ICAR-CNR, Rende (CS), Italy
Email: {cesario,cicirelli,mastroianni}@icar.cnr.it

Abstract. This paper copes with the issue of extracting mobility patterns in a urban computing scenario. The computation is parallelized by partitioning the territory into a number of regions. In each region a computing node collects data from a set of local sensors, analyzes the data and coordinates with neighbor regions to extract the mobility patterns. We propose and analyze a "local" synchronization approach, where computation regarding a specific region is performed using the information received from a subset of neighbor regions. When opposed to the usual approach, where the computation proceeds after collecting the results from all the regions, our approach offers notable benefits: reduction of computation time, real-time model extraction, better support to local decisions. The paper describes the model of local synchronization by means of a Petri net and analyzes the performance in terms of the ability of the system of keeping the pace with the data collected by sensors. The analysis is based on a real world dataset tracing the movements of taxis in the urban area of Beijing.

Keywords: smart city, mobility patterns, local synchronization, parallel computation

1 Introduction

This paper presents a novel approach that can be used for the execution of distributed smart city applications. We consider a scenario in which the parallelization of computation is performed by partitioning the territory into regions, and each region is assigned a portion of the computation, for which the input data has been collected locally. The sample application analyzed in this paper is the extraction of mobility patterns traced by people and vehicles over a city area, aiming at providing useful real-time information about mobility-related phenomena. To this purpose, we assume the existence of a network of sensors distributed in a city, which collect data about traffic, road, weather conditions, noise, etc. The analysis of such data, performed in coordination among the nodes, can provide useful real-time insights for transport users and traffic operators and can help to tackle a vast variety of mobility situations, e.g., congestion, safety, tolling.

The presented approach exploits the fact that useful information can be obtained by analyzing the data related to a local area of the territory. Specifically, if the computing node assigned to a region is able to process local data together with data coming from a subset of neighbor regions, it is possible to rapidly extract mobility patterns regarding a significant portion of the city. With this approach, the computation does not require the *all–to–all* or *global* synchronization among the nodes (i.e., all the nodes need to synchronize and collect all the data before proceeding to computation), as typical with the master-slave paradigm. Instead, the opportunity emerges of synchronizing the computation only among a limited number of parallel nodes, without the need for a central coordinator node. With this "local" synchronization, a computation at one node can proceed after being notified about the mobility patterns discovered in neighbor nodes, and can then concatenate those patterns with the ones discovered locally. This allows mobility patterns to be available much more rapidly, while the patterns regarding the whole territory can still be made available by progressively extending the area covered by the local patterns.

In a previous work, we assessed the benefits of local synchronization in a context where the objective is to predict the Internet traffic generated by mobile users in a city avenue [4][5], in a monodimensional scenario. Here we focus on a different and more general application case where the scenario is bidimensional and real-time analysis is crucial, as the computation needs to keep the pace with the production of data. We consider the case in which the computation is step-based, i.e., at the end of a predetermined time interval (i.e., an hour) it is possible to process the data regarding that time interval. The main benefits of local synchronization are:

- *Faster computation*. Local synchronization allows the overall computation to proceed faster, see Section 5. Intuitively, with global synchronization a long execution at a node compels all the other nodes to wait, thus slowing down the overall computation, while with local synchronization only the neighbor nodes (i.e., the nodes assigned to neighbor regions) need to wait before proceeding to the next step;
- *Real-time model extraction.* The previous benefit applies both to offline and online computation. In the latter case, a further advantage is that a faster computation helps to keep the pace with the data collected by sensors;
- *Better support to local decisions.* With local synchronization mobility patterns are available earlier, enabling a faster reaction to local events, e.g., a traffic congestion. This is particularly useful if the territory partitioning follows the administrative organization. For example, based on the extracted patterns, decisions on the traffic management in a city district can be performed automatically or semi-automatically;
- *Better data traffic management.* With local synchronization, data is transmitted among neighbor nodes, which is an advantage – in terms of data traffic, congestion avoidance and battery consumption in the case of wireless transmission – with respect to the case when all the data is delivered to a single node;

In this paper we focus on the first two advantages. We have modeled the local and global synchronization paradigms through a Petri net and, starting from real data extracted from a dataset regarding the mobility of users in the city of Beijing, China, we have performed a set of experiments varying the partitioning of the territory and the computational power of the nodes. We came to the conclusion that local synchronization allows the system to keep the pace with data production in a wider set of scenarios than global synchronization.

The rest of the paper is organized as follows: Section 2 discusses some related work; Section 3 describes the problem of extracting mobility patterns in an urban scenario;

Section 4 presents the Petri net that models the computation; Section 5 reports performance results regarding the ability of local synchronization to timely process the generated data; Section 6 concludes the paper and suggests some avenues for future work.

2 Related Work

The availability of urban and environmental data enables to extract mobility-related knowledge and achieve real-time traffic prediction that can support citizens in their everyday mobility. For example, it is possible to predict travel events and conditions (travel times over the street segments, traffic jams, slowed down traffic, congestion points, start-stop locations, availability of parking places) and road infrastructure conditions (bumpy road, slippery road surface, damaged road surface location).

Discovering mobility patterns from object movements is a very challenging task and several approaches to deal with it have been proposed in the literature [3, 2, 8, 9]. In [3] a sequential approach to discovery hidden periodic patterns in spatiotemporal data is proposed. In particular, authors define the spatiotemporal 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 parallel approach to estimate an object future location, based on pattern information and recent movements, is proposed in [2]; specifically, the discovered trajectory patterns are stored in the TPT (Trajectory Pattern Tree), a tree data structure exploited for an efficient and accurate prediction of future locations. In [8] the big data generated from mobile devices is analyzed in parallel at different locations and a final model is extracted by aggregating several local models following a master-worker paradigm. In particular, human mobility patterns are discovered by learning data-adaptive representations for cellular network data that are distributed across a set of interconnected nodes. In [9] a cooperative smart driving direction system is presented, 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.

3 Distributed Mobility Analysis in a Smart City

The analysis of mobility data is conceived for a scenario in which a city is partitioned into N regions, and for each region a computing node (e.g., a Raspberry unit) collects and processes data coming from the sensors located in the region. As mentioned in the introductory section, mobility patterns discovered in a region can be concatenated with those discovered in neighbor regions, so as to obtain patterns covering a wider area. The partitioning of the city in regions can follow the administrative organization of the city, i.e., the shapes of the city districts. However, for the sake of simplicity, and to allow a readier analysis of the performance results, in this paper we consider equally-sized regions, uniformly distributed over a two-dimensional grid, as represented in Figure 1.

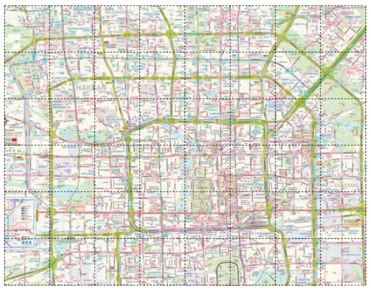

Fig. 1. A city partitioned through a bidimensional grid. This map represents the city of Beijing.

The discovery of mobility patterns is usually modeled in literature as a frequent itemset mining problem [3]. Let us suppose N sensors $s_1,...,s_N$ that collect streams of urban mobility data in a region. Specifically, each sensor s_i collect a data stream of data $D_i = \{v_i^{t_1}, v_i^{t_2}, \cdots, v_i^{t_n}, \cdots\}$, where each $v_i^{t_j}$ represents the value of a given observed measure (intensity of traffic, average speed, occupation of the lane, etc.) at the sampling time t_j. A common approach to assist mobility services is the discovery of *frequent mobility patterns* from such data. A frequent mobility pattern is represented in the form $v_i^{t_1} v_j^{t_2} \cdots v_k^{t_s}$, where the elements of the pattern represent item values that co-occur together with a high frequency (higher than a given threshold value). The mechanisms of association allow to identify the conditions that tend to occur simultaneously, or the patterns that repeat in certain conditions. As an example, a frequent pattern can represent the flow of vehicles along the city avenues, i.e., the observation that a large number of vehicles have been observed at a given location during a time interval and have been later observed in a successive time interval in adjacent locations. Moreover, rules can be derived from mobility patterns, in the form $v_i^{t_1} v_j^{t_2} \cdots \rightarrow^c v_k^{t_s}$ with time constraints $t_1 < t_2 < \ldots < t_s$. The blocks on the left and on the right are the premises and the consequence of the rule, respectively, and c is its confidence (meaning that when the premise event occurs then the consequence event will happen with probability c).

The discovery of mobility patterns has been performed by running an algorithm for frequent items and itemsets mining that we described and assessed in [1], and then by assembling the patterns discovered locally with those received by neighbor regions. The assembling operation consists in concatenating a pattern discovered in the local region with another pattern discovered in one of the adjacent regions, in the case that these two patterns overlap on the border between the two regions. In this way, two local patterns are joined and a longer inter-region pattern is discovered. This approach requires the definition of a synchronization barrier: the computation at one node, at a given step, can proceed only after receiving the results of the computation performed by neighbor nodes in the previous step. The formalization is provided in the next section.

4 The Petri Net Computational Model

The parallel computation process described in this paper is modeled by using a Petri net [6]. This formalism has been chosen because it allows to represent and analyze the

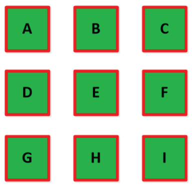

Fig. 2. Transitions modelling a grid partitioning of a city territory.

main issues related to the parallel and distributed nature of the examined scenario, in particular concerning the synchronization aspects. The city territory is partitioned into multiple regions through a bidimensional grid, and the computation step – aiming at deriving the mobility patterns – is modeled by considering a timed Petri net transition for each region. In Figure 2 we report the case in which a territory is partitioned into nine regions and, as a consequence, nine Petri net transitions are considered. The layout of the transitions mirrors the topological and neighborhood relationship among the corresponding city regions. For instance, the transition A is associated with a city region which has three neighbors, respectively modeled by the transitions B, D and E.

In the following we derive a Petri net modeling the case of local synchronization, while at the end of this section we focus on the case of global synchronization. Figure 3 shows the Petri net associated with a single region, in this case, region E, chosen here because it is the one having the largest number of neighbors in Figure 2. Beyond the transition associated with the computation, a further transition, i.e., transition P in the figure, is defined to model the data acquisition process. The acquisition process is performed through sensors that are spread over the territory. We assume that at each region this data is produced and collected at regular intervals of time, e.g., every hour, at a single node. The time interval is denoted as T_{prod}. The transition P is used to inform the computation transitions of all the regions that the data has been collected for the last time interval and is ready to be analyzed.

With reference to Figure 3, the node at region E (also referred to as node E in the following) can execute the next computation step when: (i) the node has completed the

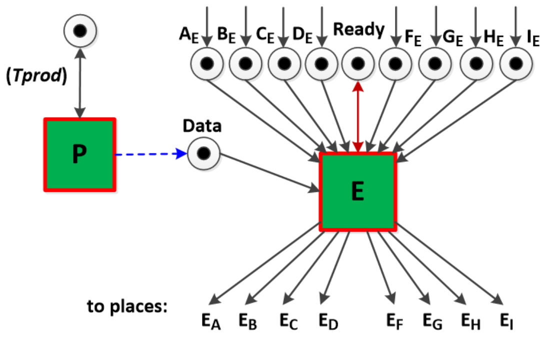

Fig. 3. Details of the Petri net model related to a single region in the city.

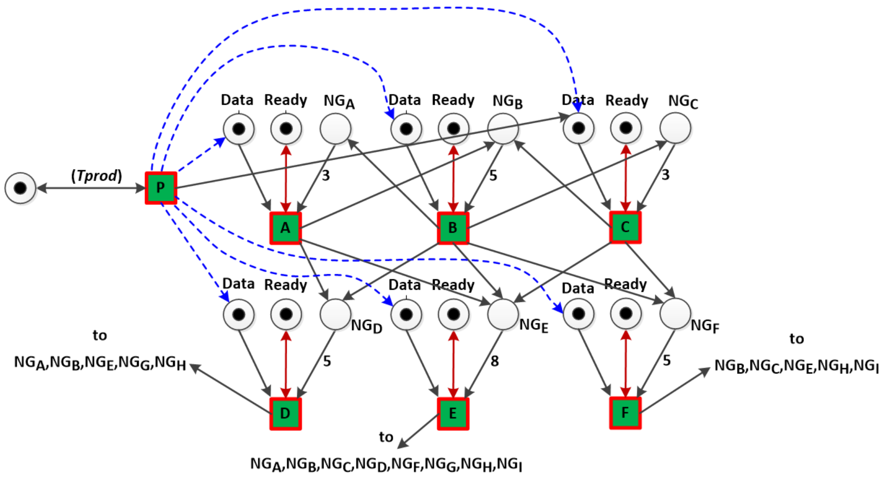
(Tprod)
P
Data
Ready
NG_A
NG_B
NG_C
A
B
C
3
5
3
NG_D
NG_E
NG_F
D
E
F
5
8
5
to NG_A,NG_B,NG_E,NG_G,NG_H
to NG_B,NG_C,NG_E,NG_H,NG_I
to NG_A,NG_B,NG_C,NG_D,NG_F,NG_G,NG_H,NG_I

Fig. 4. A Petri net of the proposed computational model.

previous computation step (ii) the node has received the results related to the previous computation at the neighbor regions and (iii) the data collected by the sensors during the last time interval has been collected and consigned to the node. In terms of the Petri net formalism, the transition E is connected by inbound arcs to ten input *places*, and in accordance with Petri net rules [7], the transition is *enabled*, and the computation can start, if all the input places hold at least one *token*. More in detail, the transition at node E is enabled when there is one token at the input place *Ready*, meaning that the previous step has been executed by node E, one token at the place *Data*, meaning that the sensor data is available, and one token at each of the eight remaining input places, meaning that all the eight neighbor nodes have completed the execution of the previous step. As an example, one token is produced at the input place A_E when the transition A has completed its execution and node A has transmitted the results to node E.

Once the node E has completed the computation step, it communicates the results to its neighbors. This is modeled by the Petri net as follows: after the firing of E, a token is consumed at each input place, and a new token is generated (i) in the place *Ready* and (ii) through the outbound arcs shown in the figure, in the input places of all the node E's neighbors, i.e., in the input place E_A of the neighbor A, in the input place E_B of the neighbor B, and so forth.

Figure 4 shows a portion of the whole Petri net model, which includes the Petri net of node E and the analogous Petri nets of the other nodes. For the sake of readability, the input places used by a node X to synchronize with its neighbors are collapsed into a single place, labeled as NG_X, and the arc that connects this place to the transition X has a weight equal to the number of X's neighbors. For example, the eight places labeled as A_E, B_E, C_E, D_E, F_E, G_E, H_E, I_E in Figure 3 are now substituted with a single node NG_E connected to node E with an arc having a weight equal to eight. Analogously, the outgoing arcs directed to the input places of neighbor nodes are substituted with a single arc on which the input places are specified[1].

From the definition of the Petri net models, it emerges that the time experienced at a generic node i at the end of the step $k+1$, denoted ad $T_i(k+1)$, is determined by the recursive expression (1):

$$T_i(k+1) = max\left(\ T_i(k), max(T_{Ngh(i)}(k)),\ T_{prod_i}(k)\ \right) +\ T_{comp_i}(k+1) \qquad (1)$$

where $T_{Ngh(i)}(k)$ is the set containing the times experienced at all the nodes that are adjacent to node i at the end of the step k, $T_{prod_i}(k)$ is the time at which the data related to step k has been consigned by the local sensors to node i, and $T_{comp_i}(k+1)$ is the time needed by node i to compute the step $k+1$.

In the case of global synchronization, the approach for synchronization is centralized: at each step a central entity receives the results (i.e., the local mobility patterns) from every node that has completed the step, and after receiving the results from all the nodes, sends an ack to every node to trigger the execution of the next step. For brevity, here we do not show the Petri net that models the global synchronization case.

5 Experimental Evaluation

In this section we analyze the performance of the sample parallel application, i.e., the prediction of mobility patterns for an urban scenario (Section 3). In this paper we focus on the computational efficiency, while we do not discuss the semantics of the extracted patterns, nor the way they are obtained by concatenating those discovered in neighbor regions, which is left to a future work.

The main objective is to analyze the advantages deriving from the local synchronization strategy, with respect to the usual global synchronization strategy. We developed a Matlab simulator that reproduces the local synchronization model, in particular the recursive expression (1) of Section 4, and the global synchronization model. The execution times of the nodes are established by considering the real execution times obtained when executing the algorithm for mobility pattern analysis on a real dataset, namely *T-Drive Trajectory Data Sample* [10]. T-Drive is a collection of GPS traces describing the movement of taxis in the city of Beijing, China. In this fashion, we were able to assess the expected performance of local and global synchronization for a real scenario. Section 5.1 describes the T-Drive dataset and the scenario of interest, while Section 5.2 reports some interesting performance results.

5.1 Dataset Description and Scenario of Interest

The temporal span of the T-Drive dataset is one week. The number of vehicles tracked is 10,357. The total distance covered by the trajectories reaches almost 9 million kilometers, with 15 millions of locations (geographic points). The total data size amounts

[1] Even with these simplifications, the Petri net maintains the desired semantics if we assume that: (i) the computation at each region has a progressive step number, (ii) each token holds the step number of the related computation, (iii) in a NG_X input place, only the tokens having the same step number can be used to enable the transition.

to about 772 MB. The original dataset was preprocessed to make it suitable for the analysis. In particular, we cleaned the data by removing all the points with unreliable position (i.e., coordinates with clear errors in latitude-longitude values) and those outside the city area of interest. The final dataset counts about 61,500 daily trajectories, each containing the set of points traced by a single taxi during a day.

The T-Drive dataset has been used to simulate a scenario where streams of data are collected by sensors distributed in the regions and analyzed to discover frequent mobility patterns in a city. As mentioned in Section 3, the analysis has been performed by running an algorithm for frequent itemsets mining that we presented in [1], and then by assembling the patterns discovered locally with those received by neighbor regions.

Since the conditions in urban environments change dynamically, the generation rate of data and the processing times vary during the day and among the regions. For this reason, we split the T-Drive data by considering time windows of one hour, and we executed the algorithm by analyzing the volume of data generated in those time windows. To analyze the statistical behavior of the algorithm execution time, we clustered the computation time by hour (for example, a cluster includes the execution times obtained from 5:00 pm to 6:00 pm of all the different days), computed the average of each cluster and then normalized each computation time with respect to the average of its cluster. The distribution of the normalized execution times is reported in Figure 5. The figure also reports the probability density function of a normal distribution with same mean and standard deviation. The two distributions appear very similar, which is confirmed by the fact that the Pearson coefficient is higher than 0.95. In conclusion, approximating the execution times with a normal curve is a reasonable assumption.

Though we base the analysis on the real data related to a specific city, in this case Beijing, we aim to assess the performance for a more general scenario, so as to draw conclusions that are related to the use of the local synchronization approach in general, and that are not tied to a given city only. To this aim, we define an experimental frame as indicated in the following:

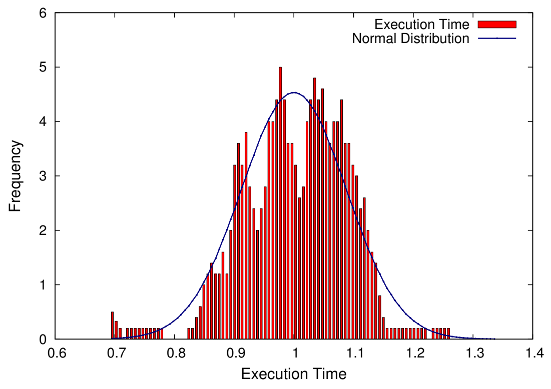

Fig. 5. Histogram of the normalized execution time of the algorithm for frequent itemsets mining. A normal distribution with same mean and standard deviation is also reported.

1. we consider a city partitioned into N regions through a bidimensional grid, with square numbers of equally sized-regions, i.e., N=2x2, 3x3, etc.
2. we assume that the extraction of mobility patterns is executed on each region by a computing node that receives and collects the mobility data produced at sensors every time interval T_{prod}, which is set to 1 hour. Therefore each computing step is associated with the computation performed on the data of a specific hour;
3. we assume that the users are distributed uniformly in the area of interest. When considering the central area of Beijing, we found this assumption reasonable. The rationale of this assumption is that the analysis of a uniform scenario is preliminary to subsequently understand what happens in a non-uniform scenario, which will be the subject of further studies;
4. all the N computing nodes are assumed to have the same computation power;
5. this computation power of nodes is varied by adopting the following approach. We assume that the average time that would be needed by a single node to perform the computation at a single step for the entire area is T_{serial}. Then, we define $R = T_{serial}/T_{prod}$ and vary the values of this ratio by varying the value of T_{serial}. Clearly the computation power of nodes in inversely proportional to the ratio R;
6. the *average computation time*, T_{node}, defined as the average time needed to perform the computation on a single node, is assumed to be proportional to the number of users located in the corresponding region, and then to the area of the region. Therefore, when the area is partitioned into N regions, $T_{node} = T_{serial}/N$. When considering the definition of R given in the previous item, it follows that $T_{node} = (R \cdot T_{prod})/N$;
7. the time needed to communicate (transmit and receive) data with the neighbor nodes is negligible with respect to the computation time. This assumption is reasonable because the nodes only need to transmit synthetic models, i.e., the results of the mobility pattern analysis, which can be done in a few seconds.

5.2 Experiments

The performance analysis was performed by using a Matlab simulator that reproduces the local and global synchronization models, as discussed at the beginning of Section 5, under the assumptions listed in the previous subsection. The local computation times used in the simulator are extracted from a normal distribution with average equal to $T_{node} = (R \cdot T_{prod})/N$ (see item 6 in Section 5.1)[2]. To analyze the benefits of local synchronization in the case that different degrees of variability are experienced, the standard deviation σ of the normal distribution was taken as a parameter, and it was set to three different values, i.e., $0.25 \cdot T_{node}$, $0.5 \cdot T_{node}$ and $1.0 \cdot T_{node}$. The number of simulated steps is equal to n_{step}. The evaluated performance indices were the following:

– the average step time T_{step}, defined as the average time to perform a computational step on all the nodes. It is computed as the time to execute n_{step} steps of the Petri net model divided by n_{step}. This index allows to assess the ability of the system to timely process the data coming from sensors. More in particular, the system does

[2] Negative values of the normal distribution are discarded and re-extracted.

not keep the pace with data production (in the following, we say that it is “unstable” for brevity) when T_{step} is larger than the acquisition interval T_{prod}, while it keeps the pace (it is “stable”) when $T_{step} = T_{prod}$. The value of T_{step} cannot be lower than T_{prod}, because the computation of a step must wait for the arrival of the related data;
– the fraction of missed deadlines, F_{miss}, defined as the fraction of times that a node receives new data coming from the sensors (the data produced during the interval T_{prod}) before completing the computation related to the previous bunch of data, i.e., the fraction of times that a single node does not keep the pace with data production.

The experiments were carried out when setting T_{prod} to one hour and the number of steps n_{step} to 720, corresponding to 30 days with the chosen value of T_{prod}. Furthermore, we considered different numbers and computation powers of nodes, more in particular, values of $N \in \{2,4,9,16,25\}$, and values of R ranging between 1 and 12.

Figure 6 shows the values of the two performance indices versus the value of R, when setting the number of nodes to 9, 16 and 25, and the value of the standard deviation σ to $0.25 \cdot T_{node}$. In the left figure we can see that the system is stable (i.e., $T_{step} = 1$ hour) when the computation is partitioned on 25 nodes. When using 16 nodes, the system is stable with local synchronization, but it is unstable with global synchronization when R is greater than 11. When using 9 nodes, we notice that there is an interval of values of R, between 6 and 7, for which the system is stable with local synchronization and unstable with global synchronization. The values of F_{miss} confirm this behavior: when the system is stable the fraction of missed deadlines is zero or negligible, while this index increases up to 1 when the system becomes unstable.

In Figures 7 and 8 we report the performance values obtained when assuming a larger variability of local computation times, as detailed in the captions. When comparing these results to those in Figure 6, we can notice two interesting phenomena:

– when the variability increases, the system tends to be unstable even with low values of R, i.e., with high values of the computation power of nodes. For example, with $R = 10$, the system is stable ($T_{step} = 1$ hour) with $\sigma = 0.25 \cdot T_{node}$ when using 16 and 25 nodes, with both local and global synchronization; it is stable with $\sigma = 0.5 \cdot T_{node}$ only when using 25 nodes, irrespective of the type of synchronization; it is stable with $\sigma = 1.0 \cdot T_{node}$ only when using 25 nodes and local synchronization;
– when the variability increases, the advantage of local synchronization increases as well. For example, if Figure 8 is observed, we can notice that there is a significant range of R values (between 8 and 10 for the case $N = 25$) for which the system is stable with local synchronization but unstable with global synchronization.

The reported results confirm the benefits brought by local synchronization. Indeed, in some scenarios with local synchronization it is possible to keep the pace with data production, while it is impossible with global synchronization and it would be required to either increase the number of nodes (which means install more computing nodes and sustain larger costs) or increase their computational power. It will be important to assess this interesting outcome when removing the simplifying assumption of uniform user distribution. When the distribution is non-uniform, it can be useful to modify the territory partitioning, for example by defining smaller regions where the user density is higher. Indeed, this is one of the issues of our current research work in this field.

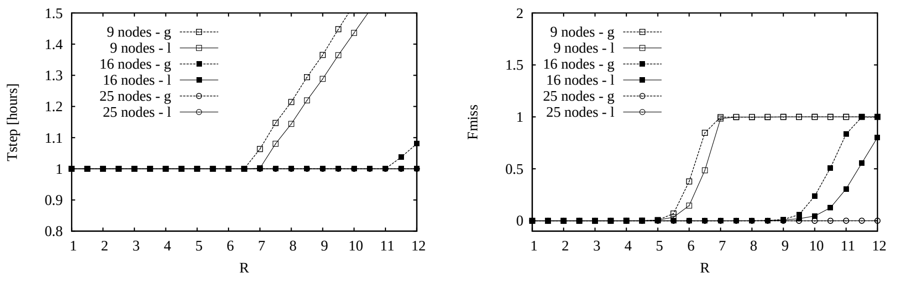

Fig. 6. Values of T_{step} and F_{miss} versus the ratio R, with $N = 9$, 16 and 25, and σ equal to $0.25 \cdot T_{node}$. Characters “g” and “l” in the legend refer to global and local synchronization, respectively.

.

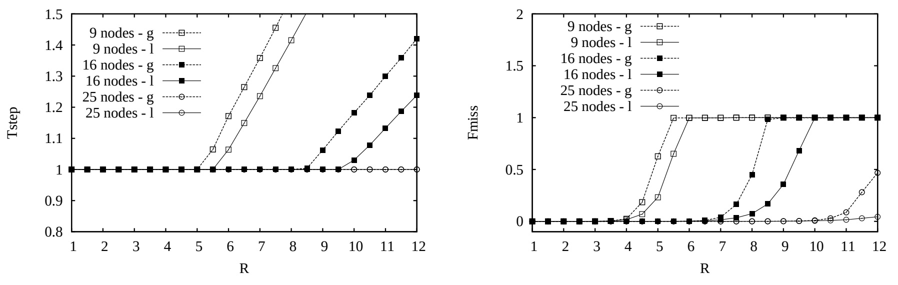

Fig. 7. Values of T_{step} and F_{miss} versus the ratio R, with $N = 9$, 16 and 25, and σ equal to $0.5 \cdot T_{node}$. Characters “g” and “l” in the legend refer to global and local synchronization, respectively.

.

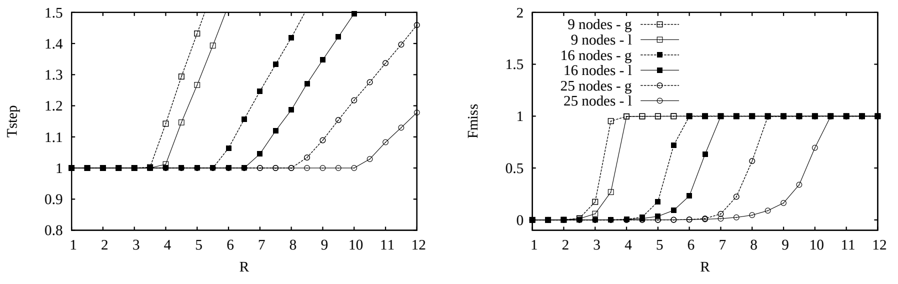

Fig. 8. Values of T_{step} and F_{miss} versus the ratio R, with $N = 9$, 16 and 25, and σ equal to $1.0 \cdot T_{node}$. Characters “g” and “l” in the legend refer to global and local synchronization, respectively. The legend is not shown in the left figure for the sake of readability.

.

6 Conclusion

In this paper, we presented an original approach based on local synchronization which is exploitable for the execution of distributed smart city applications. The main idea is to speed up the computation by limiting the overhead induced by the synchronization of the parallel nodes operating in different regions of the city. Specifically, with the presented *local* synchronization approach, each node needs to synchronize only with a set of neighbor nodes, instead of all the other nodes as required by the classical master-slave paradigm. As a specific application domain, the extraction of mobility patterns in an urban area was considered. Results, based on the analysis of a real dataset, showed that local synchronization helps to better keep the pace with the production of data in the environment, and that the advantage increases with the variability of execution times. This work has focused on the computation performance of the mobility patterns analysis. We have not discussed the semantics of the extracted patterns, nor the way they are obtained by concatenating those discovered in neighbor regions, but we intend to focus on this aspect in a future work. Other interesting research avenues are:

– extend the mobility patterns analysis to other city contexts;
– apply the approach to other smart city applications like those related to traffic management, transportation systems, and crowd monitoring and control;
– improve the approach so as to consider scenarios having a non-uniform and dynamic distribution of the workload among city regions;
– enrich the approach by furnishing a theoretical framework for the local synchronization approach.

References

1. Cesario, E., Mastroianni, C., Talia, D.: A multi-domain architecture for mining frequent items and itemsets from distributed data streams. Journal of Grid Computing 12(1), 153–168 (March 2014)
2. Jeung, H., Liu, Q., Shen, H., Tao Zhou, X.: A hybrid prediction model for moving objects. In: Proc. of the 2008 IEEE 24th Int. Conference on Data Engineering. pp. 70–79. ICDE '08, IEEE Computer Society (2008)
3. Mamoulis, N., Cao, H., Kollios, G., Hadjieleftheriou, M., Tao, Y., Cheung, D.W.: Mining, indexing, and querying historical spatiotemporal data. In: Proc. of the tenth ACM Int. Conference on Knowledge Discovery and Data Mining. pp. 236–245. KDD '04, ACM (2004)
4. Mastroianni, C., Cesario, E., Giordano, A.: Balancing speedup and accuracy in smart city parallel applications. In: Proc. of Euro-Par Workshops. Lecture Notes on Computer Science, vol. 10104, pp. 224–235. Springer, Grenoble, France (August 2016)
5. Mastroianni, C., Cesario, E., Giordano, A.: Efficient and scalable execution of smart city parallel applications. Concurrency and Computation: Practice and Experience (2017), http://dx.doi.org/10.1002/cpe.4258, Early view
6. Murata, T.: Petri nets: Properties, analysis and applications. Proceedings of the IEEE 77(4), 541–580 (1989)
7. Peterson, J.L.: Petri nets. ACM Comput. Surv. 9(3), 223–252 (September 1977)
8. Wu, T., Rustamov, R.M., Goodall, C.: Distributed learning of human mobility patterns from cellular network data. In: 2017 51st Annual Conference on Information Sciences and Systems (CISS) (2017)
9. Yuan, J., Zheng, Y., Xie, X., Sun, G.: T-drive: Enhancing driving directions with taxi drivers' intelligence. IEEE Transactions on Knowledge and Data Engineering 25(1), 220–232 (2013)
10. Yuan, J., Zheng, Y., Zhang, C., Xie, W., Xie, X., Sun, G., Huang, Y.: T-drive: driving directions based on taxi trajectories. In: Proc. of the 18th SIGSPATIAL Int. Conference on Advances in Geographic Information Systems. pp. 99–108. GIS '10, ACM (2010)