Streaming computation algorithms for spatiotemporal micromobility service availability

William Barbour*†§, Michael Wilbur*‡, Ricardo Sandoval‡, Abhishek Dubey*‡ and Daniel B. Work*†‡
*Institute for Software Integrated Systems
Vanderbilt University, Nashville, TN, USA
†Civil and Environmental Engineering
Vanderbilt University, Nashville, TN, USA
‡Electrical Engineering and Computer Science
Vanderbilt University, Nashville, TN, USA
§Email: william.w.barbour@vanderbilt.edu

***Abstract*—Location-based services and fleet management are important components of modern smart cities. However, statistical analysis with large-scale spatiotemporal data in real-time is computationally challenging and can necessitate compromise in accuracy or problem simplification. The main contribution of this work is the presentation of a stream processing approach for real-time monitoring of resource equity in spatially-aware micromobility fleets. The approach makes localized updates to resource availability as needed, instead of batch computation of availability at regular update intervals. We find that the stream processing approach can compute, on average, 62 resource availability updates in the same execution time as a single batch computation. This advantage in processing time makes continuous real-time stream processing equivalent to a batch computation performed every 15 minutes, in terms of algorithm execution time. Since the stream processing approach considers every update to the fleet in real-time, resource availability is always up-to-date and there is no compromise in terms of accuracy.**

***Keywords*-Stream processing; spatiotemporal; micromobility; resource equity**

I. Introduction

A. Motivation

The proliferation of affordable location-aware sensor technologies has made positioning applications, such as location-based services and fleet management, extremely popular. Positioning technologies such as GPS are now easily deployed in vehicles and smart phones. As cities and private companies look to optimize resources, the high velocity spatiotemporal nature of these systems present unique challenges at scale [1].

Increasingly, applications in this domain must work with dynamic data and operate in real time [2]. Allocation of city-operated microtransit (i.e., flexible route vans) and micromobility fleet management rely on constantly adapting operations based on information from these data streams [3], [4].

Large-scale spatiotemporal data of this nature is sent as a *data stream* with timestamped positions. Traditional Database Management Systems (DMBS), which are optimized for finite data sets and one-time queries, are not well suited for this type of processing. The simplest solution is to process periodically in time, as a *batch computation*. In this model, data is accumulated over a fixed time period and a batch computation is run on this data. There are two primary disadvantages to the batch processing model in this context. By waiting for a fixed period of time, stale data is introduced to downstream applications between time steps. Additionally, batch processing can require redundant computation on unchanged data.

A solution to these shortcomings is an approach that responds to new data in real-time. In this case, events from incoming data streams trigger updates to an existing model. In the case of fleet management an event could be a change in availability of a resource. Compared to a batch processing approach, which periodically checks the availability of all resources, this *stream processing* approach would avoid costly redundant computation. In a real-time application, stream processing is also more accurate because updates are computed immediately.

The main problem with stream processing approaches on real-time spatiotemporal data is designing a solution that can handle large volumes of data arriving at unpredictable rates. The cumulative cost of stream processing can also be very high if updates are frequent. Weighing batch processing and stream processing approaches in this context requires balancing trade-offs between accuracy, computation demands, and algorithmic complexity.

In this work, we focus on the problem of processing resource equity in micromobility fleets. The problem is defined as a real-time spatiotemporal process, therefore emphasizing up-to-date results. Additionally, the large, city-scale nature of the problem requires careful consideration of computational requirements.

B. Contributions and outline

The main contribution of this work is a stream processing approach for real-time monitoring of resource equity in spatially-aware micromobility fleets. The approach makes small, localized updates to resource availability as needed, instead of batch computation of availability at regular update intervals across the full study area. We find that the stream processing approach can compute, on average, 62 availability updates in the same execution time as a single batch computation. Therefore, our stream processing approach is equivalent to a batch computation performed every 15 minutes, in terms of execution time. As the stream processing model considers every update to the fleet in real-time, resource availability is always up-to-date and there is no compromise in terms of accuracy.

This article is organized as follows. Section II explores existing literature on stream computing and transportation equity. Section III formulates the stream processing model. Section IV presents a case study and its results are presented and discussed in Section V. Section VI concludes and provides specific areas of future work.

II. Related Work

A. Stream computing

Online mapping services such as Google Maps has led to significant research in real time queries on large-scale geospatial data. These applications require shortest path computation [5], [6], on massive road networks, which is computationally intensive [7], [8]. Most work in this field has focused on efficient methods for storing routing networks in traditional SQL databases [4], [9] or NoSQL databases [10]. To achieve near real-time requirements, shortest paths are pre-computed [4]. In this model, shortest paths are time-dependent and require periodic, computationally expensive batch updates [4].

Stream processing involves reactively updating a model with each new data element, rather than in large batches. Therefore, stream processing avoids the problem of stale or out-of-date data between batch updates by processing updates in real time [11]. Popular stream processing frameworks are based on directed acyclic graphs (DAG) [12], [13]. Stream processing on spatiotemporal data presents unique challenges because of the disparate spatial and temporal operators that are often available [2].

B. Transportation equity

The proliferation of micromobility services such as electric scooters and bikeshare programs has the potential to extend transportation access to groups of people beyond traditional fixed-route transit [14], [15], [16], [17]. To address equitable spatial distribution, some cities have set metrics and standards that micromobility operators must meet related to accessibility of micromobility resources [18], [19], [20], [21]. Equity requirements of this kind require real-time monitoring of fleet resources.

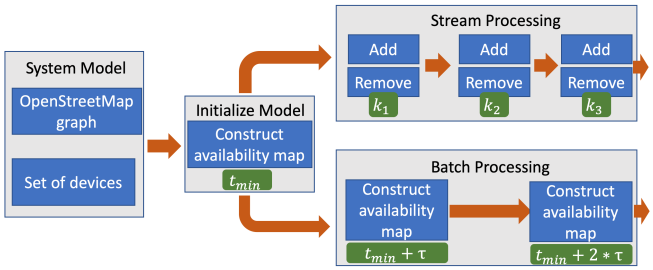

Figure 1. System models: both the stream and batch processing models start with the construction of an availability map. The stream model updates the map each time a device k is added or removed from the map. The batch processing model reconstructions the full map every τ seconds.

III. Methodology

A. Formulation of availability map

In order to quantify equity, we measure the *availability distance* of micromobility devices, defined to be the distance from a given location to the nearest available device. This will be abbreviated as the *availability* at a given point. The availability distance is measured across the city street and sidewalk network in order to emulate the actual walking distance for a user to a device. Other measures of this distance are possible, such as Euclidean distance, Manhattan distance, or a weighted form of street network distance.

An *availability map*, generally, is a map showing the availability distance at any point to the nearest micromobility device. In this work we compute the availability map in a discrete form (as an approximation of the continuous map), which is a collection of points on the city street network each with a calculated distance to the nearest available micromobility device. Using points on the city street network is an approximation of availability at residences, businesses, and other street-side locations, while having the benefit of allowing direct network routing between points on the network. Likewise, available devices are assigned to their nearest point on the street network for routing purposes.

The street network is constructed as a graph, $G = (V, E)$, which is a collection of nodes (or vertices), V, and edges, E, that enumerate valid connections between nodes. The *order* of graph G is denoted as n and is defined as the cardinality of the set of vertices, $|V|$. The difficulty of availability map construction is dependent on the order of G because this increases the number of nodes for which availability distances must be computed and, hence, the number of computations in routing through the graph.

B. Availability map algorithms

The batch and stream processing models are presented in Figure 1. Computation and updating of an availability map under the batch and streaming strategies are as follows:

- Batch: compute the availability map at the beginning of the operating day, time t_{min}, and update the map on a fixed periodic schedule every τ minutes by performing a new batch computation until the end of the operating day at t_{max}.
- Streaming: construct the full availability map at t_{min} and update portions of the map upon every removal of an available device from the network or addition of a newly available device to the network.

The set of nodes comprising the network graph is V and individual nodes are referenced as $v \in V$. Let the location of all nodes on the network graph be N such that $N \in \mathcal{R}^{nx2}$, where $n = |V|$ is number of nodes in the graph and we refer to the location of an individual node as N_v, given in latitude and longitude coordinates. Let the set of available micromobility devices at a time t be $S(t)$ and the position of each of these devices be M such that $M \in \mathcal{R}^{|S(t)|x2}$, where $|S(t)|$ is the number of devices available at time t. We refer to an individual device as $s \in S(t)$ and its location as M_s, given in latitude and longitude coordinates.

We first present the algorithm for the batch computation of the full availability map for all network nodes. This is used for periodic batch computation and to initialize the availability map at the beginning of the operating day. We then present algorithms for streaming updates to the availability map pursuant to the removal or the addition of a device on the network. For these three algorithms, we present the procedure/pseudocode for a construction or update.

1) Batch construction of full availability map:

Input: locations of any available devices, $S(t)$, at time t.

Output: for each node $v \in V$, the nearest available device, $S_v^*(t)$, and distance to device, $D_v^*(t)$.

1) Find the nearest node to each device $s \in S(t)$ based on euclidean distance between N and M, and denote this node as $v_s^* \in V$.
2) Let H be the subset of nodes (where $H \subset V$) comprised of all v_s^* at time t. For each node $h \in H$, make a queue of devices that are "located" at the node (i.e., had the same v^*) and denote the queue as $Q_h(t)$. A queue of devices is necessary because multiple devices will frequently be located at a single node. At this point, no queue is empty: $Q_h(t) \neq \emptyset \ \forall \ h \in H$.
3) Assign values $D_h^*(t) = 0$ for each $h \in H$ and set $S_h^*(t)$ to be the first device in the queue $Q_h(t)$.
4) Perform Dijkstra shortest path routing beginning with the subset H as source nodes and ending at all other nodes in V.
5) The result of the routing is the shortest path from each node $v \in V$ to any node in H (referenced to an available device). Let the value $D_v^*(t)$ be the length of this shortest path, and let $S_v^*(t)$ be the device located at the source node in H where the shortest path began. Assign the values $D_v^*(t)$ and $S_v^*(t)$ to the availability map entries of each node $v \in V$.

2) Streaming update: removal of device:

Input: a device $k \in S(t)$ to remove from the available devices at time t.

Output: updates to the availability values $S_v^*(t)$ and $D_v^*(t)$ for any affected node $v \in V$.

1) Remove k from the queue of available devices, $Q_h(t)$ at its assigned node h. If the queue contains additional devices, update any references to k in values $S_v^*(t)$ to the next device in the queue; then terminate the update.
2) If the queue does is empty after removal of k, determine the set of nodes $v \in V$ that reference device k as their nearest device (i.e., $S_v^*(t) = k$. Denote this set as I, which we refer to as *unassigned nodes* because they have now-undetermined availability values $S_v^*(t)$ and $D_v^*(t)$.
3) Construct the subset of nodes $J \subset V$, where $I \cap J = \emptyset$, as the set of nodes that are direct neighbors of any unassigned node in I. We refer to J as the *assigned neighbor nodes*.
4) Run Dijkstra routing to all unassigned nodes I, beginning with the assigned neighbor nodes J as source nodes. For each source node $j \in J$, begin the shortest path routing with the path length initialized to the availability distance $D_j^*(t)$.
5) The result of the Dijkstra routing is a shortest path from each assigned neighbor nodes $j \in J$ to any unassigned node $i \in I$. The shortest path for each node $i \in I$ has a predecessor node $j \in J$ and its new availability distance $D_i^*(t)$ is the availability distance at j, $D_j^*(t)$, plus the distance from j to i. The nearest available device for node i, $S_i^*(t)$ is the same device as that of the predecessor node j on its shortest path. The values $D_i^*(t)$ and $S_i^*(t)$ are now part of the full availability map and these update values are logged for temporal availability tracking across the day.

3) Streaming update: addition of device:

Input: a newly available device k which shall now become a part of $S(t)$, the available devices at time t.

Output: updates to the availability values $S_v^*(t)$ and $D_v^*(t)$ for any affected node $v \in V$.

1) Add device k to set of available devices $S(t)$.
2) Find the nearest node to device k, $v_k^* \in V$, by euclidean distance, and add k to the queue of available devices, $Q_{v_k^*}(t)$.
3) If $Q_{v_k^*}(t)$ was not empty before k, terminate the update.
4) Update the availability map for node v_k^* with values $D_{v_k^*}^* = 0$ and $S_{v_k^*}^* = k$.
5) Run Dijkstra routing from node v_k^* as the source to all other nodes in the graph. Explore only nodes on the routing frontier that have availability distance $D^*(t)$

less than their distance from v_k^*. That is, route only through nodes where the new device k is the nearest, because any successive nodes explored will only be further from v_k^*. Let the set of updated nodes be $C \subset V$.

6) Update the availability map with new values $D_c^*(t)$ and $S_c^*(t)$ for all $c \in C$.

C. Computation of temporal statistics

Given the ability to compute the availability distance at each node in a street network graph and the ability to update this value throughout time, we can compute the *time-weighted average availability distance* for each node. This value considers each availability distance measurement and how long each measurement persisted as a proportion of the time period. We first initialize the availability value of node v at time t_{min}. Let any successive updates to the availability value of node v occur at times $t_{1,v}$, $t_{2,v}$, etc. The final availability value of node v is thus present until the end of the analysis period at t_{max}. We denote the time values delineating availability value updates (including t_{min} and t_{max}) for node v as U_v as

$$U_v = \{t_{\text{min}}, t_{1,v}, t_{2,v}, \ldots, t_{\text{max}}\}. \tag{1}$$

For all times during the analysis period, $[t_{\text{min}}, t_{\text{max}}]$, $D_v^*(t)$ is the availability distance of node v at time t. The function $D_v^*(t)$ is a step function that changes values at times in U_v. We calculate the time-weighted average availability distance as the availability distance during each update period ($t_{i,v}$ to $t_{i+1,v}$), multiplied by the proportion of the day that time period occupied: $(t_{i+1,v} - t_{i,v})/(t_{\text{max}} - t_{\text{min}})$. The time-weighted average availability is denoted $\overline{D_v^*}$ and defined:

$$\overline{D_v^*} = \sum_{i=0}^{|U_v|-1} AD_v(t_{i,v})(t_{i+1,v} - t_{i,v}). \tag{2}$$

IV. Case Study

The case study in this work is based on data collected from micromobility operators in Nashville, Tennessee, USA. The micromobility data and the street network graph of Nashville are described first, followed by a discussion of the experimental setup for the case study.

A. Description of data

We begin data preparation with micromobility trip data from two of six major operators in Nashville between August 31, 2018, and June 21, 2019. We construct a complement to the trip dataset: all time periods during which each each micromobility device was parked between trips. We term this dataset of parking intervals the *dwell dataset*. A unique device identifier is used to determine successive trips that were taken on a device and the parking interval is the time between trips taken on the same day. Spatial coordinates (latitude and longitude) are assigned to each period of dwell between trips on the device using the endpoint of its last trip. Dwell intervals are assumed to end at the closing of each operating day at 8:00pm [22]. They begin the following day when devices are staged for pickup by users. The first dwell interval of the day is assigned a location retroactively based on the location at which its first trip of the day begins. The dwell dataset thus includes the fields: dwell start time, end time, and dwell location.

A network graph for Nashville was constructed from OpenStreetMap data, processed into undirected graph format of nodes and edges [23]. We subsequently create three versions of the network graph by partitioning graph edges to a maximum length of 25, 100, or 300 meters. These graphs are created to increase the spatial resolution of availability calculations. Numbers of nodes and edges for each partitioned network graph and the original graph are given in Table I.

Table I
Number of nodes and edges in partitioned graphs.

Partition (m)	Nodes	Edges
25	293,473	513,207
100	172,434	192,168
300	80,898	100,632
original	56,744	76,478

B. Experimental setup

A 14-week span of operating days between March 1, 2019, and June 7, 2019, were used in this case study. Each day was analyzed for 10 hours between t_{min} = 9:00am and t_{max} = 7:00pm. As described in Section III-B, the initial device availability was assessed at t_{min} by constructing the full availability map. To evaluate execution time of periodic batch processing, the availability map was re-constructed every 5 minutes until t_{max}. Streaming updates in the form of removal of a device (unavailable) or addition of a device (newly available) to the network were evaluated as they occurred chronologically between t_{min} and t_{max}. Each day in the case study was evaluated independently and execution time results for batch and streaming computations are aggregated across the day.

The algorithms are implemented in Python using NetworkX for network graphs. They were run on a computer with 16-core AMD processor running at 3.4GHz and 128GB RAM. No more than 8 simultaneous single-threaded instances of the algorithms, each evaluating separate days, were run in order to leave CPU headroom for background tasks so execution time would not be influenced.

V. Results

In this section, we first present a comparison of batch versus streaming algorithm execution time. We then compute the batch computation interval that could be realized for each

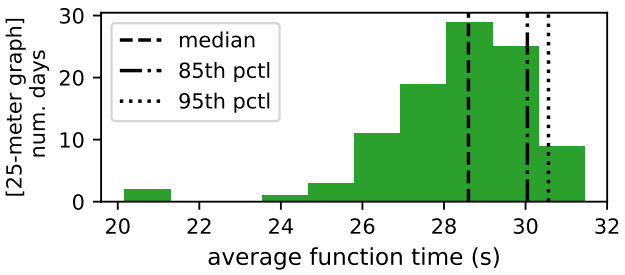

Figure 2. Distributions of daily mean execution time values for batch availability computations on the 25-meter network graph. Median, 85th percentile, 95th percentile lines are shown.

day based on the streaming computation budget. Finally, we discuss the spatial results of applying the streaming computation algorithms to compute availability statistics on a long time horizon at city scale.

A. Algorithm execution time

Each run of the batch algorithm for every 5 minutes of the dataset is naturally independent from the preceding run. The mean execution time of all iterations of the batch algorithm on a given day is computed. The distribution of daily mean execution times is shown for the 25-meter network graph in Figure 2. Lines denoting the median, 85th percentile, and 95th percentile values of the daily average execution time are shown on each subplot.

Table II shows the median and 95th percentile execution time values for the batch algorithm on each of the three partitioned graphs. As is to be expected, batch algorithm execution time is considerably longer on higher-resolution network graphs. The ranges of the distribution for the 100-meter and 300-meter graph are approximately 3 seconds and less than one second, respectively, while the 25-meter graph exhibits execution times from 20 seconds to over 30 seconds – a range of over 10 seconds.

We also measure execution time on the core routing portion of the batch algorithm function, from available devices to all nodes. This routing portion of the algorithm is identified as step (4) in the batch construction algorithm (Section III-B1). This portion of the full execution time is also shown in Table II and we see that it represents the majority of the execution time. In the cases of the 100-meter and 300-meter graphs, it is approximately 60% of execution time; but in the 25-meter graph, it is nearly 80%.

The same timing analysis was performed for the streaming algorithm. The routing component of the update algorithms (step (4) in Section III-B2 and step (5) in Section III-B3) was also timed and this portion of the execution time is included in the full function execution time. Each update – addition or removal of a device from the network – for a day of operations was executed and the days are treated independently.

Table II
BATCH ALGORITHM DAILY AVERAGE EXECUTION TIME ON EACH NETWORK GRAPH AND PERCENTAGE OF TIME SPENT IN SHORTEST PATH ROUTING.

	Batch compute execution time (s)		**% of execution time on graph routing**	
Graph	**Median**	**95th pctl.**	**Median**	**95th pctl.**
25-meter	28.6	30.56	78.4%	79.9%
100-meter	5.61	5.99	61.1%	60.9%
300-meter	2.00	2.31	56.5%	57.6%

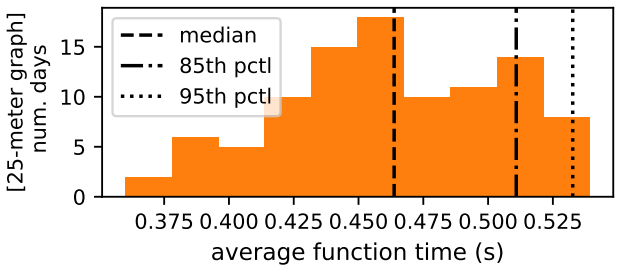

Figure 3. Distributions of daily mean execution time values for streaming availability updates on the 25-meter network graph. Median, 85th percentile, 95th percentile lines are shown.

Figure 3 shows the distribution of execution time for the streaming updates on the 25-meter graph, averaged across each day. Median, 85th percentile, and 95th percentile values are also shown by the black lines. These values are extracted in Table III for the three network graphs and it is apparent that the 100-meter and 25-meter graphs incurred execution times of nearly 3 times and 10 times greater than the execution time of the 300-meter graph, respectively. The component of the execution time representing the routing portions of the streaming update algorithms represents a very small portion of the overall function time: only 2-10%. Compared to the batch algorithm this is a much smaller portion, which is due to the other activities that the update function must perform, such as finding nodes to update, computing the neighborhood of the update, and reassigning changes. The batch algorithm, by comparison, needs only to save the result of the routing procedure in the desired format.

B. Compute time equivalence

For each day of the 14-week dataset, we have shown the mean execution time results for each version of the algorithm. Now we seek to compute the number of batch algorithm iterations that could be performed in the same computation time budget as the streaming updates take to run for the day. This quantifies the coarseness with which the periodic batch algorithm would operate given the computational budget of the streaming algorithm, which calculates availability exactly across the day. For example,

Table III
STREAMING UPDATE ALGORITHMS DAILY AVERAGE EXECUTION TIME ON EACH NETWORK GRAPH AND PERCENTAGE OF TIME SPENT IN SHORTEST PATH ROUTING.

	Stream update execution time (s)		% of execution time on graph routing	
Graph	**Median**	**95th pctl.**	**Median**	**95th pctl.**
25-meter	0.464	0.533	3.58%	7.77%
100-meter	0.133	0.159	2.71%	5.47%
300-meter	0.051	0.060	2.55%	5.50%

Table IV
LENGTH OF TIME INTERVAL BETWEEN BATCH AVAILABILITY CALCULATIONS WHEN EXECUTION TIME BUDGET IS SET AT STREAMING UPDATE ALGORITHM EXECUTION TIME FOR THE DAY.

	Compute budget (s)	Function equivalence interval (minutes)		
Graph	**Median**	**Median**	**85th pctl.**	**95th pctl.**
25-m.	1658	10.3	16.7	24.6
100-m.	506	6.5	10.9	15.8
300-m.	183	6.5	10.7	15.2

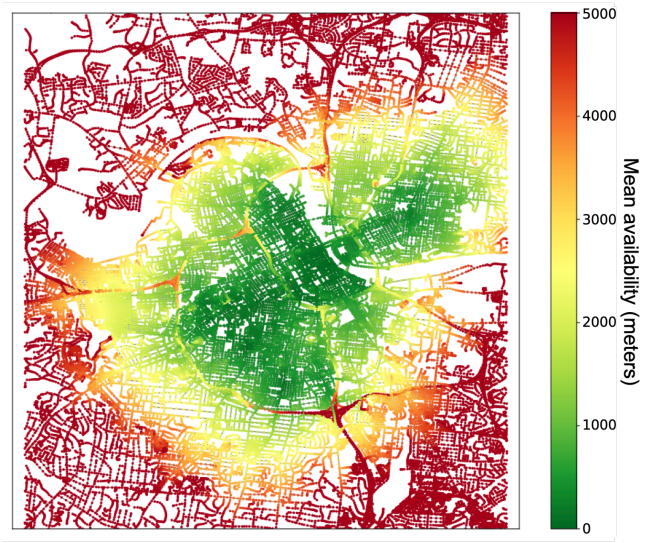

Figure 4. Mean availability at each node on the 100-meter partitioned graph across the full 14-week study period. Availability distance was capped at 5000 meters in this map.

if 2,000 updates were performed for a given day at a time of 0.10 seconds per update, then a computation budget of 200 seconds could be allocated; if each batch computation requires 5 seconds, then 40 iterations could be performed in the same budget. The coarseness of the batch algorithm is given in terms of the time interval with which the periodic iterations would be separated, given the compute budget. Following the same example with 40 batch iterations equivalent to the total stream update execution time, these iterations could be spread out across the 10-hour day used in the experiments at 15-minute intervals.

Table IV shows the computational equivalence intervals for the batch algorithm under the three partitioning sizes of the network graph, as well as the average daily execution time budget. The 100-meter and 300-meter graphs exhibit a nearly identical distribution of interval times, with the median time intervals of 6.5 minutes and extending up to 15 minutes in the 95th percentile case. This is not to say that the total computational budget under the two graph versions is the same. In fact, the median budget was 506 seconds for the 100-meter graph and 183 seconds for the 300-meter graph. The 25-meter graph, however, demonstrated a much longer batch computation interval that would be allowed, with a median of 10 minutes and 95th percentile value of nearly 25 minutes. Its daily median execution time was 1658 seconds. This highlights the larger disparity in execution times between batch and stream algorithms under the 25-meter graph than the other two graphs.

C. Long-term availability

We now give an example of the long-term city-scale spatiotemporal statistics that can be computed efficiently with specialized streaming update algorithms. The time-mean availability at each node is computed for each day in the 14-week dataset, as described in Section III-C. The daily mean values are then averaged across the 98 days in the case study and result in the aggregate availability map, at the 100-meter partitioned level, in Figure 4. The map shows very smooth distribution of availability, minus a few anomalies due to highway road segments and the river running through the city. This map required an execution time of 797 minutes (13 hours, 17 minutes) to compute 433,518 streaming updates. This execution time included the batch computation that is requited at the beginning of each day to initialize the availability map before streaming updates can begin.

VI. CONCLUSION

In this work we presented a stream processing approach for real-time monitoring of resource equity in spatially aware micromobility fleets. Empirical results from a case study of scooter micromobility data in Nashville, TN show that the computational cost of our streaming approach is equivalent to a batch processing model executed at 15 minute intervals. Future work on this topic includes extending our approach to other fleets including bus transit systems, and adapting our approach to a decentralized implementation on fog networks. Additional algorithmic enhancements could further improve the computational advantage for stream processing.

ACKNOWLEDGEMENTS

The authors would like to thank the Metropolitan Gov. of Nashville and Davidson County, TN, for their ongoing support. These analyses were derived from information

collected by Metro and shared with Vanderbilt as permitted by current legislation through the ongoing partnership. This material is based upon work supported by the National Science Foundation under Grant No. CMMI-1853913.

REFERENCES

[1] P. Huang and B. Yuan, "Mining massive-scale spatiotemporal trajectories in parallel: A survey," in *Trends and Applications in Knowledge Discovery and Data Mining*. Springer, 2015, pp. 41–52.

[2] S. Eom, S. Shin, and K.-H. Lee, "Spatiotemporal query processing for semantic data stream," in *Proceedings of the 2015 IEEE 9th International Conference on Semantic Computing (IEEE ICSC 2015)*. IEEE, 2015, pp. 290–297.

[3] M. A. B. C. B. Sethu and R. E. Katibah, "Spatio-temporal stream processing in microsoft streaminsight," *Data Engineering*, vol. 69, 2010.

[4] J. Sankaranarayanan and H. Samet, "Roads belong in databases," *Data Engineering*, p. 4, 2010.

[5] E. W. Dijkstra *et al.*, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[6] N. Jing, Y.-W. Huang, and E. A. Rundensteiner, "Hierarchical encoded path views for path query processing: An optimal model and its performance evaluation," *IEEE transactions on knowledge and data engineering*, vol. 10, no. 3, pp. 409–432, 1998.

[7] Z. Chen, H. T. Shen, X. Zhou, and J. X. Yu, "Monitoring path nearest neighbor in road networks," in *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, 2009, pp. 591–602.

[8] W. Zeng and R. L. Church, "Finding shortest paths on real road networks: the case for a*," *International journal of geographical information science*, vol. 23, no. 4, pp. 531–543, 2009.

[9] R. Obe and L. Hsu, "Postgis in action," *GEOInformatics*, vol. 14, no. 8, 2011.

[10] M. Miler, D. Medak, and D. Odobašić, "The shortest path algorithm performance comparison in graph and relational database on a transportation network," *Promet-Traffic&Transportation*, vol. 26, no. 1, pp. 75–82, 2014.

[11] M. Cherniack, H. Balakrishnan, M. Balazinska, D. Carney, U. Cetintemel, Y. Xing, and S. B. Zdonik, "Scalable distributed stream processing," in *CIDR*, vol. 3, 2003, pp. 257–268.

[12] M. H. Iqbal and T. R. Soomro, "Big data analysis: Apache storm perspective," *International journal of computer trends and technology*, vol. 19, no. 1, pp. 9–14, 2015.

[13] S. Kulkarni, N. Bhagat, M. Fu, V. Kedigehalli, C. Kellogg, S. Mittal, J. M. Patel, K. Ramasamy, and S. Taneja, "Twitter heron: Stream processing at scale," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, 2015, pp. 239–250.

[14] S. Groth, "Multimodal divide: Reproduction of transport poverty in smart mobility trends," *Transportation Research Part A: Policy and Practice*, vol. 125, pp. 56–71, 2019.

[15] K. Lucas, G. Mattioli, E. Verlinghieri, and A. Guzman, "Transport poverty and its adverse social consequences," in *Proceedings of the institution of civil engineers-transport*, vol. 169, no. 6. Thomas Telford (ICE Publishing), 2016, pp. 353–365.

[16] H. Titheridge, R. Mackett, N. Christie, D. Oviedo Hernández, and R. Ye, "Transport and poverty: a review of the evidence," 2014.

[17] D. Booth, L. Hanmer, and E. Lovell, "Poverty and transport," *A report prepared for the World Bank and DFID. London: Overseas Development Institute*, 2000.

[18] SFMTA, "Stationless bikeshare permit application distribution guidelines."

[19] S. Adrian Leung, "Bikeshare."

[20] San Francisco Municipal Transportation Agency, "SFMTA stationless bikeshare program permit application."

[21] M. G. of Nashville and D. County, "Second substitute bill bl2018-1202(as amended)," *Metropolitan Code of Laws*, vol. Title 12, 2018.

[22] G. McKenzie, "Spatiotemporal comparative analysis of scooter-share and bike-share usage patterns in Washington, D.C." *Journal of Transport Geography*, vol. 78, pp. 19–28, 2019.

[23] G. Boeing, "Osmnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks," *Computers, Environment and Urban Systems*, vol. 65, pp. 126–139, 2017.