Using Stream Processing to Find Suitable Rides: An Exploration based on New York City Taxi Data

Roswita Tschümperlin[1], Dominik Bucher[1], and Joram Schito[1]

1 **Institute of Cartography and Geoinformation, ETH Zurich, Stefano-Franscini-Platz 5, CH-8093 Zurich, Switzerland**
troswita@ethz.ch, dobucher@ethz.ch, jschito@ethz.ch

Abstract

Changes in the mobility landscape, such as the emergence of shared or autonomous mobility, or the increasing use of mobility as a service, require an adaption of information technologies to satisfy travelers' demands. For example, the deployment of autonomous taxis will necessitate systems that automatically match people with available cars. Combined with the enormous increase in available data about transport systems and the people using them, mobility providers have to think about how to build flexible and scalable systems that satisfy their customers' information demands. In our work we explore how streaming frameworks (as part of Big Data infrastructures) can be used to process mobility data, and how the spatiality of the underlying problem can be used to improve the performance of the resulting systems. We currently experiment with different approaches on how to build a mobility data stream processing pipeline, and evaluate the different approaches against each other in terms of scalability, ease of use, and possiblity to integrate with data from other mobility providers.

1 Introduction

Mobility and transport must and already started to undergo large transformations due to sustainability goals set by many countries, new technological advances and the saturation of existing transport systems (3). Technological advances are not restricted to fields such as electric or autonomous cars, but also cover an increasing miniaturization of sensors and computing technology, which leads to enormous amounts of data about transport systems and the people using them. Several trends, including the emergence of shared and autonomous mobility, lead to new requirements for information technology (IT) to support travelers (6). In this work we look at taxis, which are often regarded as prime candidates for replacement with autonomous cars in the near future and will require automated IT to match people with taxis.

The use of Big Data streaming frameworks makes it possible to build systems that automatically scale (depending on the problem size), and makes them more easily deployable compared to having to build a distributed mobility processing framework from scratch. We currently experiment with different architectures for processing taxi data and evaluate them in terms of scalability, ease of use, and their suitability for integration with data from external providers (e.g., environmental data). In the work presented here, we are focusing on finding the nearest taxis for any client request (essentially solving a nearest neighbor problem), and discuss several important additions to the streaming system required for future autonomous taxi matching.

In a geographical context, Big Data frameworks were recently used to process location-based social network data, street network data and mobility and movement trajectories (e.g., (5)). There is only little public research on using *streaming* (in contrast to "plain" Big Data) frameworks to process geographical data: For example, Domoney et al. (2) look at the processing of sensor data streams, while Maarala et al. (4) use streaming to process and analyze traffic data. We argue that there is a lack of best practices when employing streaming frameworks to process spatio-temporal data, in particular with regards to traffic and mobility, an aspect that will be increasingly important for future forms of mobility. With this work, we show how spatial

Spatial big data and machine learning in GIScience, Workshop at GIScience 2018, Melbourne, Australia.
Editors: Martin Raubal, Shaowen Wang, Mengyu Guo, David Jonietz, Peter Kiefer

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

indexing structures can be used for mobility data, and hope to initiate a discussion on the crucial components within a streaming pipeline for matching autonomous taxis.

The two best-known approaches used by Big Data streaming frameworks are *microbatching* and *true continuous streaming* (7). In microbatching, data is aggregated into smaller batches (usually of several hundred milliseconds) which are then sequentially processed. As taxi matching only has to be near real-time and vehicle and client data have to be *joined*, we are using Apache Spark[1], which adopted a microbatching approach in order to reuse its Hadoop-based structures for streaming. In addition to supporting well-known operations of Big Data systems (such as *map*, *reduce*, *join*, etc., which are automatically distributed among worker nodes) Spark Streaming offers functionality for stateful streams (e.g., to continuously update the position of a taxi) and checkpointing (for fault recovery), both essential to process mobility data in a realistic setting.

2 Method

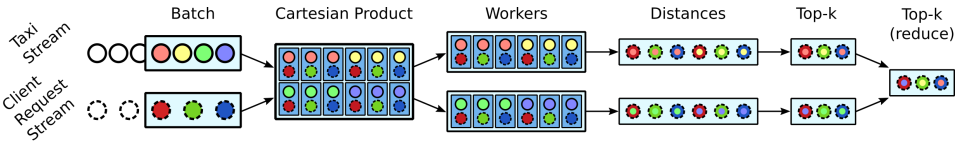
We present two approaches how to match a stream of taxi data with incoming and pending transport requests (i.e., to find the closest k taxis to an incoming client request). Both the taxi data and client request streams consist of a tunable number of updates per second, sent as JSON-encoded packets over a WebSocket connection. Each packet of the incoming taxi data stream contains the *id* of the taxi it refers to (one out of several thousand), its current *location (longitude, latitude)*, the *number of people* currently traveling in it, and the *next destination* (if known). On the other hand, the stream of incoming client requests consists of the *origin (longitude, latitude)* and the *destination* of a desired journey. It is beneficial to maintain a *state* of the locations, as otherwise only the information within a single batch would be available. Technically, it would suffice to only either maintain the state for the taxis or the client requests, but this would inevitably lead to longer delays before finding the optimal match (e.g., if solely taxi locations were stateful, they could only be matched with client requests whenever those appear in the stream, and not when at a later point in time updated taxi locations would lead to a better match).

Figure 1 shows two different streaming pipelines evaluated as part of this research. Distributed processing of big data can easily be done if the individual records resp. the computations performed on them are independent. An evident approach to compute the k nearest taxis for a number of client requests is to simply combine all taxis with all requests, compute the distance between each pair, sort them according to this distance, and take the top k. Figure 1a shows this approach, which does not use any spatiality inherent to the problem. Optimally, in order to compute the top-k matches, one would not have to compute the distances between all taxis and client requests. Spatial indexing structures, such as R-trees (1), can help reducing the number of computations that need to be performed. In the approach shown in Figure 1b, we build an index for each batch of taxi data, and use these indexes to reduce the number of computations performed on each worker (the workers get a subset of client requests for which they have to find the closest taxis).

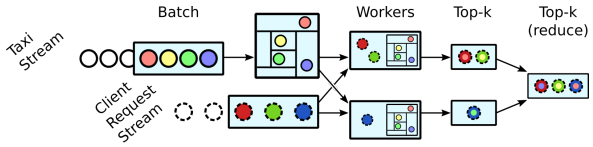
3 Data and Experiment

We are currently performing experiments based on real taxi data from New York City[2]. Because this dataset only contains dropoff and pickup coordinates, we used the Open Source Routing

[1] See spark.apache.org for details.
[2] The NYC taxi data is publically available starting from 2009 and can be downloaded from www.nyc.gov/html/tlc/html/about/trip_record_data.shtml.



(a) The *cartesian pipeline*: Incoming taxi updates and client requests are processed in batches of several hundred milliseconds. The cartesian product is computed for all taxis and client requests in a (stateful) batch. These tuples are distributed on an adjustable number of workers which compute the distances between taxis and clients. Each worker then takes the top-k pairs, and uses the streaming framework to perform a reduction step, where ultimately the top-k taxis for each request remain.



(b) The *separated KNN pipeline*: On the master (resp. in the driver program), a spatial index is built for the taxis. This index is then sent to the workers, which use it to minimize the number of computations needed per client request.

Figure 1 The two different mobility stream processing pipelines analyzed within this work.

Machine (OSRM)[3] to compute exact trajectories. As the dataset does not contain any indication about which taxi served a particular request, and what taxis did between two served clients, we used a heuristical approach to create a daily driving schedule (without gaps) for each taxi. A streaming application developed in Go[4] constantly reads the taxi schedules and outputs packets with an adjustable frequency to WebSocket streams. The streaming application that tests the different pipelines is thus able to receive taxi and client request streams in a controlled fashion (with respect to throughput, number of taxis and clients, time window, etc.).

Figure 2 shows the influence of the taxi and client request throughputs on the time that is required to process a single batch (normalized by the number of taxis; the batch size is 500 ms). On the one hand, the processing time decreases with an increasing number of taxi updates per second as the overhead of the streaming framework is spread across more taxi updates. On the other hand, an increase in client requests leads to an increase in processing time, as proportionally more computations have to be performed. The effects of an increase of the client requests are less prominent for the separated KNN pipeline, as the spatial index reduces the computational complexity at the expense of having to build it for every batch.

4 Discussion and Future Work

The first results presented in this work show that it is possible and likely beneficial to use a streaming framework for finding potential matches between taxis and the travelers requesting them. However, it is still unclear what kind of pipeline architecture will perform well, in particular when considering the integration of external services and data providers. Our findings suggest using the spatiality inherent in the problem of finding the nearest taxis can greatly reduce the computational efforts, leading to a speedup in the order of one magnitude in our case.

The use of realistic taxi data in combination with a tuneable streaming source allowed us to quickly test various streaming pipelines in a reproducible setting. As next step we planned a

[3] The Open Source Routing Machine (OSRM) can be downloaded from project-osrm.org.
[4] Go is a systems programming language developed by Google, see golang.org.

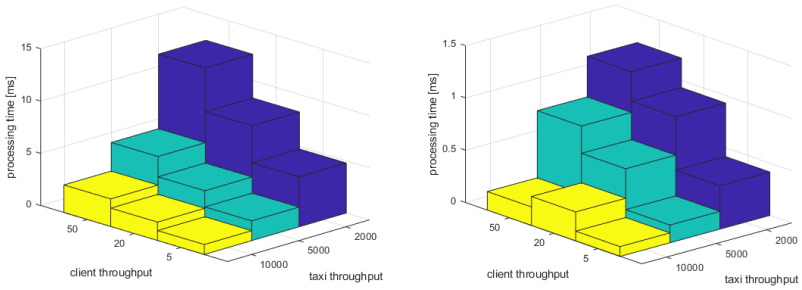


Figure 2 Normalized processing times of the two pipelines presented in Figure 1 (cartesian pipeline on the left, separated KNN on the right).

more in-depth examination of the scalability of various architectures, e.g., by varying the number of workers in the cluster, by changing the partitioning (of work resp. data), and by employing a reusable spatial index that can be updated with new positions in every batch. In addition, we here omitted the integration of other services (e.g., OSRM for exact routing). Integrating such a service is particularly problematic as it can lead to enormous (and unpredictable) delays which in turn lead to instabilities in the streaming framework. Finally, in line with the goal of enabling people to share their taxis, we work on an addition to the streaming pipeline that considers potentially occupied taxis and lets taxis pick up multiple passengers. Especially the latter part requires a more complex approach, as solely solving the nearest neighbor problem often will not lead to a globally optimal solution.

Acknowledgements. This research was supported by the Swiss National Science Foundation (SNF) within NRP 71 “Managing energy consumption” and is part of the Swiss Competence Center for Energy Research SCCER Mobility of the Swiss Innovation Agency Innosuisse.

References

1 Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The r*-tree: an efficient and robust access method for points and rectangles. In *Acm Sigmod Record*, volume 19, pages 322–331. Acm, 1990.

2 W Frank Domoney, Naseem Ramli, Salma Alarefi, and Stuart D Walker. Smart city solutions to water management using self-powered, low-cost, water sensors and apache spark data aggregation. In *Renewable and Sustainable Energy Conference (IRSEC), 2015 3rd International*, pages 1–4. IEEE, 2015.

3 Moshe Givoni and David Banister. *Moving towards low carbon mobility*. Edward Elgar Publishing, 2013.

4 Altti Ilari Maarala, Mika Rautiainen, Miikka Salmi, Susanna Pirttikangas, and Jukka Riekki. Low latency analytics for streaming traffic data with apache spark. In *Big Data (Big Data), 2015 IEEE International Conference on*, pages 2855–2858. IEEE, 2015.

5 Paras Mehta, Christian Windolf, and Agnès Voisard. Spatio-temporal hotspot computation on apache spark (gis cup). In *24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 2016.

6 Harvey J Miller. Beyond sharing: cultivating cooperative transportation systems through geographic information science. *Journal of Transport Geography*, 31:296–308, 2013.

7 Maninder Pal Singh, Mohammad A Hoque, and Sasu Tarkoma. Analysis of systems to process massive data stream. *CoRR, abs/1605.09021*, 2016.