TrajParquet: A Trajectory-Oriented Column File Format for Mobility Data Lakes

Nikolaos Koutroumanis
Department of Digital Systems
University of Piraeus
Piraeus, Greece
koutroumanis@unipi.gr

Christos Doulkeridis
Department of Digital Systems
University of Piraeus
Piraeus, Greece
cdoulk@unipi.gr

Chiara Renso
Inst. of Inf. Sci. and Technologies
National Research Council of Italy
Pisa, Italy
chiara.renso@isti.cnr.it

Mirco Nanni
Inst. of Inf. Sci. and Technologies
National Research Council of Italy
Pisa, Italy
mirco.nanni@isti.cnr.it

Raffaele Perego
Inst. of Inf. Sci. and Technologies
National Research Council of Italy
Pisa, Italy
raffaele.perego@isti.cnr.it

ABSTRACT

Columnar data formats, such as Apache Parquet, are increasingly popular nowadays for scalable data storage and querying data lakes, due to compressed storage and efficient data access via data skipping. However, when applied to spatial or spatio-temporal data, advanced solutions are required to go beyond pruning over single attributes and towards multidimensional pruning. Even though there exist solutions for geospatial data, such as GeoParquet and SpatialParquet, they fall short when applied to trajectory data (sequences of spatio-temporal positions). In this paper, we propose TrajParquet, a format for columnar storage of trajectory data, which is highly efficient and scalable. Also, we present a query processing algorithm that supports spatio-temporal range queries over TrajParquet. We evaluate TrajParquet using real-world data sets and in comparison with extensions of GeoParquet and SpatialParquet, suitable for handling spatio-temporal data.

CCS CONCEPTS

• **Information systems** → **Data access methods**; **Data layout**.

KEYWORDS

Apache Parquet, spatio-temporal, trajectories, mobility data

ACM Reference Format:
Nikolaos Koutroumanis, Christos Doulkeridis, Chiara Renso, Mirco Nanni, and Raffaele Perego. 2023. TrajParquet: A Trajectory-Oriented Column File Format for Mobility Data Lakes. In *The 31st ACM International Conference on Advances in Geographic Information Systems (SIGSPATIAL '23), November 13–16, 2023, Hamburg, Germany*. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3589132.3625623

SIGSPATIAL '23, November 13–16, 2023, Hamburg, Germany

ACM ISBN 979-8-4007-0168-9/23/11.
https://doi.org/10.1145/3589132.3625623

1 INTRODUCTION

In the era of big data, cloud-based storage and querying data in raw [1] or in minimally-processed formats in data lakes is becoming increasingly popular [10], as it alleviates the need for time-consuming ETL (Extract-Transform-Load) operations and it reduces the data-to-query time. This is particularly evident in domains that handle big and complex data, such as scientific, social, and mobility data. In the context of data lakes [4], specific data formats have been proposed to fit with this new infrastructure for data management. In particular, column-oriented data formats, such as Parquet [9], inspired by Google's Dremel [2], are widely used, due to salient features that include compressed storage and efficient data access via data skipping [8].

Although columnar formats are efficient for tabular data, they cannot be trivially applied to more complex data types, such as geospatial data. Thus, researchers have proposed extensions, such as SpatialParquet [7], while the Open Geospatial Consortium (OGC) makes an effort to standardize GeoParquet. However, these formats are suitable for static geospatial data, whereas many modern applications produce large volumes of dynamic, time-evolving spatial data, such as trajectories of moving objects.

Motivated by this need, in this paper, we introduce *TrajParquet*, a columnar storage format designed for scalable storage and querying of trajectory data that extends Apache Parquet. In Section 2, we describe the representation of trajectory data in this format, followed by a practical indexing approach that allows data skipping at query time. In Section 3, we perform a comparative evaluation of TrajParquet against extensions of SpatialParquet and GeoParquet (to handle the temporal dimension) over two real-world data sets. The results demonstrate that our proposal offers significant performance improvements. In Section 4, we conclude the paper and discuss future work.

2 TRAJPARQUET

2.1 The TrajParquet Format

The TrajParquet structure format is designed for the storage of trajectories as sequences of spatio-temporal positions capturing the movement and location of objects or entities over time. A single trajectory may be composed of thousands positions, which can

```
message TrajectorySegment {
    required BINARY entityId;
    required INT64 segmentId;
    required BINARY valsX;
    required BINARY valsY;
    required BINARY valsT;
    required DOUBLE minX;
    required DOUBLE maxX;
    required DOUBLE minY;
    required DOUBLE maxY;
    required INT64 minT;
    required INT64 maxT;
}
```

Listing 1: TrajParquet message structure format.

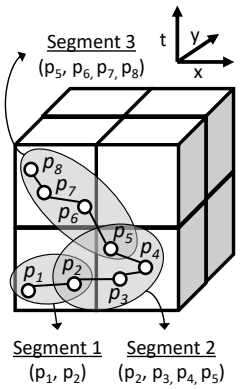


Figure 1: Trajectory segmentation.

be difficult to store and query it as a single unit. For this reason, typically, a trajectory is represented as sequences of *segments* that contain a number of successive spatio-temporal positions. Two successive spatio-temporal positions constitute a *line segment*. Listing 1 shows the message structure of a `TrajectorySegment` in the TrajParquet format. The values of spatio-temporal positions (longitude, latitude, timestamps) are stored separately in three binary fields (`valsX`, `valsY`, and `valsT`), one for each dimension. Each field is represented as a byte array that contains all the consecutive values for a specific dimension, corresponding to the consecutive positions of a moving entity. Moreover, we store the spatio-temporal bounding box of the trajectory in six fields (`minX`, `maxX`, `minY`, `maxY`, `minT`, `maxT`). Additionally, a TrajParquet message holds the entity's identifier in a binary field (`entityId`) and the identifier of the trajectory segment in a 64-bit signed integer field `segmentId`.

2.2 Trajectory Segmentation and Grouping

Given an input data set of trajectories, the first step is to generate the corresponding TrajParquet files. However, depending on the use-case scenario, trajectories may consist of a high number of positions. For instance, a vessel trajectory may consist of thousands of positions in the case of transatlantic trips or it may correspond to multiple routes between two ports along with the stops between routes [12]. To solve this challenge, the common approach followed in the literature is to perform *trajectory segmentation*, essentially splitting a trajectory into multiple segments and storing these as individual objects [5]. As a result of the segmentation process, a trajectory can be represented as a sequence of successive `TrajectorySegments`. Another challenge is how to group together `TrajectorySegments` in the same data page of TrajParquet. Clearly, if this *grouping of segments* is performed at random, then the result would be data pages that span large spatial areas and long time intervals, thus offering limited opportunities for skipping pages. Instead, it is beneficial to group together `TrajectorySegments` with high spatial vicinity and with similar time span, so as to form compact data pages possibly increasing the opportunities for data skipping.

To address these two challenges, we propose to partition the 3D spatio-temporal domain using the Hilbert space-filling curve (SFC) [3]. Notably, as depicted in Fig. 1, the space is partitioned in 3D non-overlapping cubes, each covering a specific spatial area for a certain time span. Then, the Hilbert SFC is used to assign 1D numeric values to cubes, in a way that preserves data locality, i.e., neighboring cubes are expected to have close identifier values. In this way, grouping is performed based on the 1D values assigned by the SFC, by means of a sorting operation. Therefore, trajectory segments that are located in neighboring cubes will be stored on the same data page with high probability.

The actual trajectory segments are formed based on containment in a specific cube. In Fig. 1, the trajectory of an object is depicted that consists of positions: $\{p_1, p_2, \ldots, p_8\}$, and it is split into 3 segments. Note that when we generate the trajectory segments, we introduce some data replication in order to preserve the connection of line segments between successive trajectory segments. Specifically, we add in each segment the last spatio-temporal point of the previous segment of the trajectory. For example, using this technique, the trajectory segments in Fig. 1 are: $\{p_1, p_2\}$, $\{p_2, p_3, p_4, p_5\}$, and $\{p_5, p_6, p_7, p_8\}$. By doing so, we ensure that all of the line segments that cross two or more cubes will be represented in the trajectory's segments as two successive points, without being missed. This, guarantees the correctness during the processing of spatio-temporal range queries, since all of the line segments that compose a trajectory are represented in the end.

2.3 Query Processing

In this section, we describe the algorithm for processing spatio-temporal range queries. A spatio-temporal range query q is defined as a box with lower and upper bounds $q = [(x_l, y_l, t_l), (x_u, y_u, t_u)]$. The query fetches all of the spatio-temporal line segments that intersect with the box. The algorithm follows the *filter-and-refinement* paradigm, where a set of candidate objects are retrieved efficiently (filtering), followed by a process that excludes the false positives, i.e., those candidate objects that are not query results (refinement).

Filtering Phase. The boundaries of the spatio-temporal range are expressed for each dimension as interval constraints. This is done for those fields that relate to the values of its Minimal Bounding Box (MBB). The constraints are combined under a logical *and* operator, passed as a predicate for push-down filtering. The execution of the filter fetches all messages, i.e., trajectory segments, whose intervals in the longitude, latitude and temporal dimensions intersect with the query. It should be noted that only a limited set of data pages are accessed from disk. This is due to the application of data skipping. Thus, the filtering phase is the one that is associated with disk accesses. The subsequent processing (refinement phase) occurs in the main memory.

Refinement Phase. Having fetched a set of trajectory segments whose MBBs intersect with the query, an additional procedure follows for discarding the false positive spatio-temporal lines of segments, i.e., the ones that do not intersect with the query. Each line (two subsequent points) for each trajectory segment is checked for intersection with the query. Lines that do not intersect with the query, are disregarded from the trajectory segments.

After the refinement phase, a concatenation procedure follows. The concatenation procedure connects consecutive trajectory segments into a single trajectory object. Clearly, the applied trajectory

segmentation may split a trajectory into more than one segment (messages) if it crosses more than one cubes. In this manner, we manage to have a result set of consecutive trajectories.

2.4 Extensions of Existing Spatial Formats

In this section, we explain how existing Parquet formats for geospatial data could be extended to support the temporal dimension. These extensions will serve as baselines to compare them against our proposal TrajParquet.

GeoParquet[1] is an incubating Open Geospatial Consortium (OGC) standard for storing geospatial data in Parquet. Each message handles a geometry in a well-known binary (WKB) format in a binary field. Along with the geometry, the coordinate values of the lower and upper bounds of its MBB are stored in separate 64-bit floating point fields. GeoParquet enables push-down spatial filtering by exploiting the MBB. To handle trajectories, we extend the Geoparquet format, referring to it as *GeoParquet+Time*. In this format, each message corresponds to a trajectory segment and stores a linestring in WKB format. We add a repeated 64-bit signed integer field that keeps the temporal values of each location. In addition, we add two fields to keep the temporal interval of this segment. In this way, the existing spatial bounding rectangle is enhanced with the temporal dimension, enabling push-down spatio-temporal filtering. Concerning the information of the moving entity, we add a binary field that holds the identifier of the object and a 64-bit signed integer field that holds the segment identifier.

SpatialParquet [7] is a Parquet extension that supports geospatial data types. Coordinate values are stored in double fields under nested repeated groups. In this way, several geometric types can be represented. The geometry type is indicated by a specific value, stored in a 32-bit signed integer field in the message. We extend SpatialParquet to a trajectory-oriented format, referring to it as *SpatialParquet+Time*. In this format, each message represents a trajectory segment. Thus, we add a linestring to each message, which represents, in the repeated nested coordinates group, the consecutive locations of the moving entity. To cover the temporal values of the spatio-temporal traces, we also add a 64-bit signed integer field alongside the values of the coordinates. Similarly to the GeoParquet extension, we add a binary field that holds the entity's identifier and a 64-bit signed integer field that holds the segment identifier.

3 EXPERIMENTS

3.1 Experimental Setup

The experimental evaluation is conducted by evaluating TrajParquet, against the GeoParquet+Time and the SpatialParquet+Time, trajectory-oriented extensions, denoted with GP+Time and SP+Time, respectively. To ensure a fair comparison, the three formats are evaluated under the same trajectory segmentation and grouping approach, presented in Section 2.2. They also utilize the query processing algorithm portrayed in Section 2.3. However, for the SpatialParquet+Time approach, the filtering phase is not in effect, as it cannot exploit the push-down filtering procedure. This is a known limitation of the current Parquet library, and it is related to repeated fields.

[1]https://geoparquet.org

The assessment is based on the performance of spatio-temporal range queries, defined by ranges in the three dimensions. A range query retrieves all trajectory segments that intersect with the 3D spatio-temporal box defined by the query.

Data Sets. We evaluate the efficiency of the Parquet formats on two real-world trajectory data, Geolife and Brest. Geolife [11] is a GPS trajectory data set which contains 26M spatio-temporal positions from 187 users in the time span of 64 months. Brest[2] is an integrated and curated maritime data set that contains vessel positions, weather data, contextual data, etc. [6]. We use the 19M vessel positions of a period of 6 months to reconstruct trajectories for 4, 349 vessels.

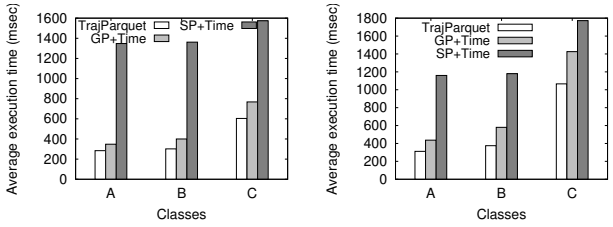
Query Generation. We generate 1, 000 box queries of equal length in every dimension located at random locations of the data space. We separate the queries in 3 classes (A, B and C) based on the size of the query result set. Class A covers the queries with the smallest number of retrieved results (up to the 20th percentile), while class C covers the queries with the largest number of retrieved results (larger than the 80th percentile). Class B contains the queries that fall between the 40th and the 60th percentiles. For each class, we report the average execution time.

Platform and Software. The experiments were executed on a machine equipped with 3.6GHZ Intel core i7-4790 processor, 16GB DDR3 1600MHz RAM, 500GB SSD disk drive and Ubuntu 22.04.2 LTS operating system. We implemented the TrajParquet format in Java, based on the Parquet Java library. For the GP+Time and SP+Time formats, we extended the GeoParquet and SpatialParquet message structure format of the Java implementation that was used in the SpatialParquet paper [7]. The experiments run via the Apache Spark framework v 3.0.1, utilizing the resilient distributed dataset (RDD) API interface. The parquet Hadoop integration package (*parquet-hadoop*) is used for reading and writing parquet files from the RDD interface. The default configuration settings are used for the Apache Parquet storage format, while for the execution environment of Apache Spark, the parameter that is set exclusively is the driver memory, 4GB for the pre-processing and 2GB for querying procedure.

Pre-processing. A pre-processing procedure is applied to store the Geolife and Brest data sets in Parquet format. We utilize the SNAPPY compression for all of the Parquet format structures. For all of the structures (TrajParquet, GP+Time and SP+Time), the pre-processing procedure takes the same time for each data set, 120 sec for Geolife and 90 sec for Brest. Even though we exploit the Hilbert space-filling curve when partitioning the space, we use a different representation precision for each data set. The more bits we use, the larger the number of cubes covering the data set's space. A different precision is needed as there are differences in the size of the spatio-temporal space the various data sets cover. For the Geolife data set we use 8 bits that result in 16,777,216 partitions, while for the Brest data set we use 5 bits for a total of 32,768 partitions.

Code availability. The code of TrajParquet and the baselines used for the experiments is publicly available[3] to favor the reproducibility of the results and to allow improvements and extensions from the scientific community.

[2]Publicly available at https://zenodo.org/record/1167595
[3]https://github.com/nkoutroumanis/TrajParquet/



(a) Geolife **(b) Brest**

Figure 2: Average query execution time on the Geolife and Brest data set.

3.2 Experimental Results

Fig. 2 shows the average execution time of the different Parquet formats per query class for the Geolife and Brest data set. The first observation is that TrajParquet outperforms all the competitor approaches in all cases. GP+time is the second best, while SP+time performs worse than the other two approaches in all the tests because it does not perform predicate push-down for the spatio-temporal constraint due to the use of the repeated field. This shows that the format proposed in SpatialParquet cannot be extended to handle sequences of spatio-temporal positions effectively.

In Fig. 2a, for the Geolife data set, the gain in performance for the TrajParquet format compared to the best-performing competitor (GP+Time) is 18.3%, 24.5% and 21.3% for classes A, B and C, respectively. In Fig. 2b, for the Brest data set, the biggest gain in performance is observed, 28.6%, 35.3% and 25.2% for classes A, B and C, respectively.

Table 1: Size (in MB) of data sets for TrajParquet and the different extensions GP+Time and SP+Time.

Data set	TrajParquet	GP+Time	SP+Time
Geolife	424	428	424
Brest	255	263	219

Table 1 shows the sizes of the data sets (in MB) for each Parquet format, where SNAPPY compression is applied. The SP+Time approach achieves the most efficient compression compared to the other approaches for the Brest data set. For the Geolife data set, the TrajParquet and the SP+Time approaches have the same memory footprint, smaller than the GP+Time approach.

Discussion. We infer from the experimental evaluation that neither GP+Time nor the SP+Time can handle trajectory data efficiently. The GP+Time format stores linestrings in a well-known binary (WKB) format, requiring deserialization when the points of the linestrings are parsed. Also, the timestamp values are stored in a repeated field, and thus they are fetched one by one in a streaming fashion. This slows down the execution of queries.

On the other hand, SP+Time format stores all of the elements (x, y, t) of the linestring in repeated fields. Thus, it cannot perform data skipping, as it handles the values that are queried in three different repeated fields. Instead, the TrajParquet format does not face any of these problems. The spatio-temporal points of the line segments are stored in three separate binary fields where each one constitutes a byte buffer. The buffer handles contiguous sequences of the bytes of the actual values in a dimension. When a push-down predicate filtering is applied, each byte buffer is fetched as a single value, and then the actual (sequential) values are read.

4 CONCLUSIONS

In this paper, we propose an extension of Apache Parquet, called TrajParquet, suitable for representing and storing trajectory data at scale. The proposed format relies on a technique for trajectory segmentation and a method for grouping spatio-temporal positions to generate compact data pages that increase the opportunity for data skipping. We present an algorithm for range queries over trajectory data that outperforms adaptations of existing approaches in Parquet for spatial data. Our comparative experimental study using two real-life data sets demonstrates the advantages of our approach.

Many directions for future work can be identified. One obvious direction is providing efficient support for additional query types, such as k-nearest neighbors. Then, creating more compact data pages by exploiting sophisticated grouping methods that do not sacrifice correctness is of interest.

ACKNOWLEDGMENTS

The research leading to the results presented in this paper has received funding from the European Union's funded Projects SoBigData++ and MobiSpaces under grant agreements no 871042 and no 101070279 respectively.

REFERENCES

[1] Ioannis Alagiannis, Renata Borovica, Miguel Branco, Stratos Idreos, and Anastasia Ailamaki. 2012. NoDB: Efficient query execution on raw data files. In *Proc. of SIGMOD*. ACM, 241–252.
[2] Sergey Melnik, Andrey Gubarev, Jing Jing Long, Geoffrey Romer, Shiva Shivakumar, Matt Tolton, and Theo Vassilakis. 2010. Dremel: Interactive Analysis of Web-Scale Datasets. *Proc. VLDB Endow.* 3, 1 (2010), 330–339.
[3] Bongki Moon, H. V. Jagadish, Christos Faloutsos, and Joel H. Saltz. 2001. Analysis of the Clustering Properties of the Hilbert Space-Filling Curve. *IEEE Trans. Knowl. Data Eng.* 13, 1 (2001), 124–141.
[4] Fatemeh Nargesian, Erkang Zhu, Renée J. Miller, Ken Q. Pu, and Patricia C. Arocena. 2019. Data Lake Management: Challenges and Opportunities. *Proc. VLDB Endow.* 12, 12 (2019), 1986–1989.
[5] Costas Panagiotakis, Nikos Pelekis, Ioannis Kopanakis, Emmanuel Ramasso, and Yannis Theodoridis. 2012. Segmentation and Sampling of Moving Object Trajectories Based on Representativeness. *IEEE Trans. Knowl. Data Eng.* 24, 7 (2012), 1328–1343.
[6] Cyril Ray, Richard Dréo, Elena Camossi, Anne-Laure Jousselme, and Clément Iphar. 2019. Heterogeneous integrated dataset for Maritime Intelligence, surveillance, and reconnaissance. *Data in Brief* 25 (2019), 104141.
[7] Majid Saeedan and Ahmed Eldawy. 2022. Spatial Parquet: A column file format for geospatial data lakes. In *Proc. of SIGSPATIAL*. ACM, 102:1–102:4.
[8] Paula Ta-Shma, Guy Khazma, Gal Lushi, and Oshrit Feder. 2020. Extensible Data Skipping. In *Proc. of IEEE BigData*. 372–382.
[9] Deepak Vohra. 2016. *Apache Parquet*. 325–335.
[10] Grisha Weintraub, Ehud Gudes, and Shlomi Dolev. 2021. Needle in a haystack queries in cloud data lakes. In *Proc. EDBT/ICDT Workshops (CEUR Workshop Proceedings, Vol. 2841)*. CEUR-WS.org.
[11] Yu Zheng, Xing Xie, and Wei-Ying Ma. 2010. GeoLife: A Collaborative Social Networking Service among User, Location and Trajectory. *IEEE Data Eng. Bull.* 33, 2 (2010), 32–39.
[12] Dimitris Zissis, Konstantinos Chatzikokolakis, Giannis Spiliopoulos, and Marios Vodas. 2020. A Distributed Spatial Method for Modeling Maritime Routes. *IEEE Access* 8 (2020), 47556–47568.