A Python-Based Framework for Preprocessing and Vehicle Flow Analysis of ETC2.0 Probe Data for Efficient Data Handling

Rena Kato, Souma Noguchi, Ahmad Altaweel, Haruki Sato, Guanyu Su and Hiroaki Wagatsuma*
Graduate School of Life Science and Systems Engineering, Kyushu Institute of Technology, 2-4 Hibikino, Wakamatsu-ku, Kitakyushu, 808-0196, Japan

Email: kato.rena844@mail.kyutech.jp, noguchi.souma288@mail.kyutech.jp, altaweel.ahmad770@mail.kyutech.jp, sato.haruki103@mail.kyutech.jp, guanyu3275@mail.kyutech.jp, waga@brain.kyutech.ac.jp
*Corresponding Author

Abstract

According to the advancement of the recording systems embedded in road infrastructure, large-scale traffic datasets are available for government-related organizations and academic-industrial frameworks. The conventional scheme of software design, such as cascaded processing, has a limitation in the speed and capacity of data processing. For such big-data analysis, a big-data–oriented software design is necessary. In the present study, we present a preliminary Python-based framework for efficient data processing and analysis of ETC2.0 probe data. The framework enables streamlined handling of large-scale traffic datasets and supports vehicle-level identification with origin–destination specifications, useful for analyzing the mechanisms of traffic jam formation and Zone 30 safety near schools. Designed for flexible adaptation to target regions and historical periods, it provides a reproducible and open workflow to facilitate future research on traffic flow and ITS applications.

Keywords: ETC2.0 probe data, Vehicle flow analysis, Traffic congestion mechanism, Intelligent Transport Systems (ITS)

1. Introduction

In recent years, the utilization of traffic big data has attracted increasing attention in the context of research in the field of intelligent transportation systems (ITS). ETC 2.0 probe data is known for the high-frequency collection of vehicle positions, speeds, accelerations/decelerations, and travel events through bidirectional communication between onboard units (OBUs) and roadside units (RSUs), allowing detailed analysis of vehicle behavior under real traffic conditions [1][2]. ETC 2.0 data has been discussed as a key resource for analyzing congestion factors on highways [3], extracting traffic accumulation patterns around tourist areas [4], and constructing route choice models based on travel history information [5]. At the same time, micro-level traffic analyses, such as lane-level trajectory estimation [6] and modeling of signalized intersection crossing behavior [7][8], have raised high expectations for effective road design to reduce traffic jams and road accidents.

However, most existing studies assume offline statistical analysis, and the complex data formats and analysis units (DRM-link or census unit) require substantial effort for preprocessing, time-series synchronization, and map-based visualization. Although the data specifications provided by the National Institute for Land and Infrastructure Management (NILIM) are becoming publicly available [9], a reusable and general-purpose programming framework for researchers is highly expected yet remains under development.

In the present study, we address the challenge of developing a Python-based system for analyzing ETC 2.0 probe data. The proposed framework can efficiently load data in different formats (e.g., 1-1, 1-2, 2-1 to 2-7) [10][11], synchronize, analyze, and visualize vehicle-level travel histories, and is designed to support extensions for traffic behavior analysis in specific regions. It will contribute to future road design in the Munakata City area.

2. Methodology

2.1. Overview: Dask DataFrame Assembly

This study employs Dask DataFrames as the primary data-processing engine to efficiently manage a large collection of compressed files [10][11][12]. Conventional single-threaded processing with Pandas becomes impractical due to memory constraints and the sequential nature of I/O operations [13]. Dask's capability for parallel computation, out-of-core processing, and lazy execution enables scalable assembly of a unified dataset while minimizing memory consumption [10][12][14]. The following subsections describe the design and implementation of this pipeline.

2.2. Construction of Delayed Tasks

If NILIM datasets are provided as a set of zipped (ZIP) files to reduce data transmission costs, each ZIP file can be processed using *dask.delayed*, which wraps the file-loading function into a deferred computation [10][12]. Instead of executing the function immediately, each invocation generates a node in a directed acyclic graph (DAG) representing the entire workflow. This mechanism allows thousands of ZIP files to be represented as lightweight computational tasks before any data is read into memory, enabling scalable parallel processing and efficient out-of-core computation [10][14].

2.3. Integration via dd.from_delayed()

The collection of delayed tasks is combined into a single logical dataset using dd.from_delayed() [10][11]. In this stage, Dask constructs a partitioned DataFrame, where each partition corresponds to the output of an individual delayed ZIP-loading task. Interestingly, the unified DataFrame functions only as a logical structure that references the DAG [10][12], meaning that the underlying data is not yet loaded or materialized in RAM. This design enables efficient manipulation of very large datasets that would otherwise exceed memory capacity through out-of-core and parallel execution mechanisms [10][14].

2.4. Lazy Execution Mechanism

As discussed above, all operations—including file extraction, parsing, concatenation, and type conversion—remain unexecuted until an explicit computation trigger such as the *.compute()* or *.to_parquet()* methods is invoked [10][12]. This approach is called lazy execution, in which Dask postpones evaluation until the final stage of the pipeline. By deferring work, Dask avoids constructing intermediate DataFrames, thereby reducing both memory pressure and I/O overhead [10][14]. Once execution is triggered, Dask's scheduler optimizes and executes the DAG across available CPU cores [10][12]. Each partition is processed independently and in parallel, enabling efficient utilization of multi-core environments. Intermediate results are released from memory as soon as they are no longer required, which prevents the accumulation of large temporary objects [10][14].

2.5. Data Type Optimization

For the optimization of data preservation, data types are refined to reduce memory consumption within each partition before the partitions are combined [11][13]. For high-cardinality string features, categorical encoding can be effectively applied, while numeric columns are downcast to the smallest feasible integer or floating-point type [13][15]. Designing these optimizations at the partition level ensures that the data remain lightweight throughout the pipeline [10][11]. These optimizations can be considered to reduce the overall RAM usage of the resulting Dask DataFrame [10][14]. By shrinking the in-memory representation of each partition, a sufficient number of partitions can be processed concurrently, which improves both throughput and scalability.

2.6. Rationale for Using Dask

The introduction of Dask is key to leveraging parallel processing for accelerating the ingestion and transformation of large file collections. Parallel processing not only shortens processing time, but also enables scaling of pipelines by utilizing additional CPU resources when implementing the same code on larger systems. Dask processes data on a partition-by-partition basis, allowing efficient handling of datasets that exceed the system's physical memory and preventing memory exhaustion that can occur in Pandas workflows. Dask's lazy execution and partitioned DataFrame structure work seamlessly with columnar storage formats such as Parquet. As mentioned above, the final write operation can achieve parallel output and efficient compression thanks to Dask's optimized execution graph [12][16].

3. Proposed Framework

3.1. ETC2.0 Probe Data Processing Framework Using Dask and GeoParquet

If NILIM datasets are provided as a set of zipped (ZIP) files to reduce data transmission costs, each ZIP file can be processed using *dask.delayed*, which wraps the file-loading function into a deferred computation [10][12]. Instead of executing the function immediately, each invocation generates a node in a directed acyclic graph (DAG) representing the entire workflow. This mechanism allows thousands of ZIP files to be represented as lightweight computational tasks before any data is read into memory, enabling scalable parallel processing and efficient out-of-core computation [10][14].

3.2. Raw Data Acquisition

ETC2.0 probe measurements are collected by roadside units (RSUs) deployed across Japan. Each RSU periodically transmits a ZIP archive containing vehicle probe logs. File names embed key metadata such as:

- Acquisition date (YYYYMMDD),
- Travel origin and destination (U/D)
- Traveling data as the set of GPS locations.

3.3. Parallel Ingestion and Metadata Extraction Using Dask Delayed

Each ZIP archive is processed in parallel through a Dask delayed task. For each file, the following steps are executed:

1. Metadata extraction: date, location ("place"), lane type, and travel direction are extracted from the filename using regular expressions.

2. Temporary extraction: the embedded data.csv is unpacked into a temporary directory.
3. CSV parsing: the CSV file is read using pandas, and the previously extracted metadata are assigned as additional columns.
4. Cleanup: temporary files are deleted immediately to minimize disk usage.

Since each of these steps is wrapped as a delayed function, they contribute to the global task graph without triggering immediate execution.

3.4. Dask DataFrame Assembly (Lazy Execution)

All delayed parsing tasks are aggregated into a unified Dask DataFrame through dd.from_delayed(). This integration does not load the data into memory; instead, Dask constructs a partitioned logical DataFrame whose partitions correspond to individual ZIP archives.

3.5. Spatial and Time Zone Filtering

Prior to trajectory synthesis, the DataFrame can be filtered to target a specific study area and time zone. If the timestamp (*GPStime_dt*) associated with the GPS location of a vehicle is already stored in the original data and converted to datetime64 type (if it is represented as a string, the to_datetime method needs to be applied) as a column in the Pandas DataFrame (ddf in this case). The following assignment is then introduced:

```
import dask.dataframe as dd

ddf_points = ddf_points.assign(
    hour = ddf_points["GPStime_dt"].dt.hour
)

# Spatial Filtering
ddf_points = ddf_points[ddf_points["location"] ==
"Munakata City"]

# Time Zone Filtering
morning = ddf_points["hour"].between(7, 9)
# 7:00, 8:00, 9:00 are concatinated
evening = ddf_points["hour"].between(17, 19)
# 17:00, 18:00, 19:00 are concatinated

ddf_points = ddf_points[morning | evening]
```

Due to Dask's lazy execution model, such filtering only modifies the task graph; no computation is executed until the final materialization stage.

3.6. Trajectory Reconstruction via Dask Shuffle and Partition-Level Processing

A distributed shuffle (sorting) by trip identifier (accounting for the possibility that the same vehicle may pass the same location multiple times in a single day) is initially performed, and vehicle trajectories are reconstructed from all point records grouped by Vehicle ID and Trip Identifier. In the partition-level LineString (connection of GPS locations) construction, trajectory synthesis is performed using *map_partitions*, where each partition is processed with Pandas or GeoPandas:

1. Points are sorted by timestamp (*GPS_time_dt*) and index order..
2. Coordinate tuples (longitude, latitude) are assembled.
3. A LineString geometry is constructed from the ordered coordinate sequence.
4. Each trip is returned as a single record in a partition-level GeoDataFrame

The result is a structured dataset in which each row represents a fully reconstructed trip.

3.7. Trajectory Reconstruction via Dask Shuffle and Partition-Level Processing

The reconstructed trajectories are written to disk in **GeoParquet** format.
Each Dask partition is exported independently, yielding a directory structure such as:

```
geo_parquet_trip_lines/
    trip_lines_0000.parquet
    trip_lines_0001.parquet
...
```

All files comply with the OGC GeoParquet specification and store, such as LineString geometries, trip identifiers, RSU-derived metadata, temporal attributes. Parallel output can reduces I/O time and facilitates downstream spatial analytics.

3.8. Integration with GIS and Large-Scale Geo-Analytics Tools

The resulting GeoParquet dataset is compatible with a wide range of geospatial systems such as QGIS.

By exporting trajectories to an scalable geospatial format, the framework ensures efficient distribution and long-term reproducibility of large-scale probe data analyses.

4. Results and Discussion

For the benchmark test comparing traditional Pandas-based data processing with Dask-based processing, we used open data from the City of Chicago data portal, specifically the Taxi Trips dataset covering the years 2013 to 2023 [17]. This dataset was artificially transformed into a format consistent with the ETC 2.0 specification, in which each data file is provided as a ZIP archive. The contents of the Taxi Trips dataset were realigned to match the ETC 2.0 data schema and then compressed as ZIP files to create ETC 2.0–mimicked data. Using this transformed dataset, the computational performance does not differ

significantly for datasets on the order of 100 files. Performance differences are expected to emerge when processing datasets on the order of 1,000,000 to 10,000,000 records or higher. (Figure 1).

Instead of a purely probabilistic HMM-based map-matching, we propose a physics- and geometry-based trajectory reconstruction approach. Sparse ETC2.0 probe points (7–10 s interval) are first deterministically snapped to a road network using distance, heading, and speed consistency. Then, physically feasible paths between consecutive observations are selected by minimizing a cost function combining travel time consistency, curvature penalization, and speed constraints. Finally, piecewise-constant or piecewise-constant-acceleration motion models interpolate along the selected road paths, yielding dense trajectories on the road geometry (Figure 2).

For each ETC 2.0 probe measurement (e.g. longitude, latitude, speed, brake), nearby candidate links are evaluated using a deterministic cost function:

$$C = w_d d_\perp^2 + w_\theta(\Delta\theta)^2 + w_v(v_{obs} - v_{lim})^2$$

where $d_\perp$ is lateral distance, $\Delta\theta$ is heading difference, v_{obs} is observed speed, and v_{lim} is the link's speed limit. The link minimizing C gives longitudinal coordinate $\boldsymbol{s_i}$. For successive observations *i* and *i+1*, the expected speed on a candidate path is: $v_k = L_k/\Delta t$. The mean observed speed is: $\hat{v}_i = (v_i + v_{i+1})/2$. Path cost is computed as: $J_k = \alpha(v_k - \hat{v}_i)^2 + \beta \sum_j \phi(\Delta\theta_j)$.

Then, continuous trajectory reconstruction can be reproduced as:

- Constant-speed interpolation is performed using: $s(t) = s_i + v_i\,(t - t_i)$.
- Corrected speed under geometric constraints: $v^* = (s_{i+1} - s_i)/(t_{i+1} - t_i)$.
- Constant-acceleration interpolation uses: $s(t) = s_i + v_i\,(t - t_i) + 1/2\,a_i\,(t - t_i)^2$.

```
======================================
      BENCHMARK START
======================================
[pandas] load time: 0.34 sec
[pandas] shape: (1000, 31)

#zip files found = 102
[Dask] load + compute time: 1.15 sec

[Dask] ddf columns: ['trip_id', 'taxi_id',
'trip_start_timestamp', ...]...
```

Figure 1: THe benchmark test for 102 samples.

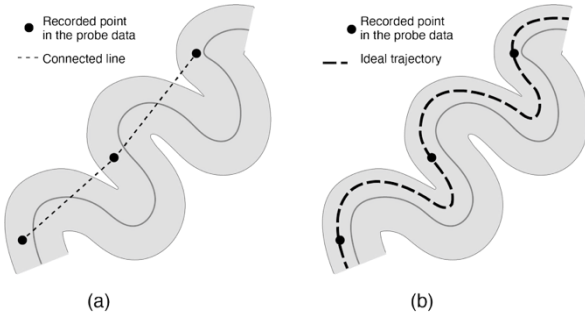

Figure 2: The schematic illustration of the result of the reconstructed trajectory of a vehicle by using the proposed method. (a) simple connection of probe data positions and (b) ideal reconstructed trajectory.

5. Conclusion

The proposed framework provides an end-to-end solution for ETC2.0 probe data, combining Dask's parallel and lazily evaluated computational model with GeoParquet's efficient geospatial storage. The architecture supports nation-wide datasets, minimizes memory usage, enables fast trajectory reconstruction, and preserves high interoperability with modern GIS and analytical ecosystems.

Acknowledgements

This work was supported in part by JSPS KAKENHI (JP17H06383, JP24K07387) the research grant from Munakata City. The authors would like to express their sincere gratitude to the Traffic Engineering and ITS Research Group at the National Institute for Land and Infrastructure Management (NILIM) for offering invaluable guidance on data specifications on ETC2.0 probe data

References

1. ITS Division, Road Traffic Department, National Institute for Land and Infrastructure Management, Ministry of Land, Infrastructure (NILIM), Transport and Tourism (MLIT Japan), ETC2.0: "ETC2.0" Probe information.
2. Cabinet Office, Government of Japan, Smart Mobility, ETC 2.0: Two-Way Communication between Vehicles and the Road, March 2021.
3. S. Hirai, J. Xing, S. Kai, R. Horiguchi and N. Uno, Analysis on the Resting Behavior of Inter-urban Expressway Users with ETC2.0 Probe Data, JSTE Journal of Traffic Engineering Special Edition A (Research Paper) 3(4), 2017, pp. A_36–A_45.
4. T. Miyoshi, E. Hasegawa and S. Tanaka, Trial Analysis of the Usage of Rest Area by Using ETC2.0 Probe Data, Journal of Transportation Engineering 3(2) (Special Issue B), 2017, pp. B_6–B_12.
5. H. Makino, S. Itsubo and A. Goto, A Study of the Utilization of ETC2.0 Probe Data for Policy Evaluation of Road Administration, Policy and Practice Studies 3(1), 2017, pp. 15–30.
6. R. Imai, H. Inoue, K. Nakamura, Y. Yamamoto, Y. Tsukada, K. Ikemoto and N. Namba, Consideration of Driving Lane Estimation Using ETC2.0 Probe Data, JSTE

Journal of Traffic Engineering Special Edition A (Research Paper) 10(1), 2024, pp. A_259–A_264.
7. Y. Murano, N. Matsuda, K. Yokochi, S. Satouchi, J. Tanabe, T. Maekawa and K. Fukuda, Validation of the Effectiveness of the Bottleneck Point Method Using ETC 2.0 Probe Data Through Its Application to Various Congestion Situations, The 62th Annual Meeting of Japan Society of Civil Engineers, 2020, ID 62-46-01.
8. R. Yamamoto, H. Seya, M. Tomari, D. Murakami and S. Yasuda, Identification of Queueing Areas for Signalized Intersections Using Vehicle Trajectory Data, Japanese Journal of JSCE 80(12), 2024, Article ID: 24-00109.
9. New Road Technology Conference, Proposal for the Basic Design of Next-generation ETC Through Utilization and Evaluation of ETC2.0 Data, Report on the Results of Research and Development on Technologies Contributing to the Improvement of the Quality of Road Policies 31(1), 2022.
10. Dask Developers, Dask Documentation – Parallel Computing with Task Scheduling.
11. Dask Developers, Dask DataFrame: Scalable Analytics in Python.
12. M. Rocklin, Dask: Parallel Computation with Blocked Algorithms and Task Scheduling, Proceedings of the 14th Python in Science Conference (SciPy), 2015.
13. NumFOCUS, Inc., Pandas Documentation – Python Data Analysis Library, Version 2.3.3, 2025.
14. Dask Developers, Dask Working Notes,
15. W. McKinney, Data Structures for Statistical Computing in Python, Proceedings of the 9th Python in Science Conference, 2010.
16. J. Shi and J. Lu, Performance Models of Data Parallel DAG Workflows for Large Scale Data Analytics, Distributed and Parallel Databases 41, 2023, pp. 299–329.
17. Chicago data portal, Taxi Trips ,2013-2023.

Authors Introduction

Ms. Rena Kato

She received her associate's degree from the Department of Electrical and Electronic Systems Engineering, National Institute of Technology, Toyota College, Japan in 2024. She is currently enrolled in the Department of Systems Design and Informatics, Faculty of Computer Science and Systems Engineering, Kyushu Institute of Technology, Japan.

Mr. Soua Noguchi

He enrolled in the Intelligent Control Course, Department of Mechanical and Intelligent Systems Engineering, at the Kyushu Institute of Technology (Kyutech) in 2022. He is currently a fourth-year undergraduate student at the same university, conducting research on traffic congestion mechanisms using ETC 2.0 probe data at the Wagatsuma Laboratory.

Mr. Ahmad Altaweel

He received his Bachelor's degree in Electronics and Communication Engineering, ALBAATH University, Syria. He is currently a Ph.D. student at Kyushu Institute of Technology, Japan.

Mr. Haruki Sato

He earned a bachelor's degree from the Faculty of Engineering at Kyushu Institute of Technology (Kyutech) in Japan in 2025. He is currently a master's student at Kyushu Institute of Technology.

Mr. Guanyu Su

He is a first-year master's student at the Graduate School of Life Science and Systems Engineering, Kyushu Institute of Technology (Kyutech), Japan. He received his bachelor's degree from Southwest Minzu University, China, in 2023. His research focuses on embedding risk-related semantic information into Lanelet2-based high-precision maps and analyzing their contribution to autonomous driving safety.

Dr. Hiroaki Wagatsuma

He received his M.S., and Ph.D. degrees from Tokyo Denki University, Japan, in 1997 and 2005, respectively. In 2009, he joined Kyushu Institute of Technology, where he is currently an Associate Professor of the Department of Human Intelligence Systems. His research interests include non- linear dynamics and robotics. He is a member of IEEE.