



PDF Download
3748636.3766538.pdf
09 January 2026
Total Citations: 0
Total Downloads: 160

 Latest updates: <https://dl.acm.org/doi/10.1145/3748636.3766538>

SHORT-PAPER

sparkmobility: A Spark-based Python Library for Processing, Modeling, and Analyzing Large Mobility Datasets

SHANGQING CAO, University of California, Berkeley, Berkeley, CA, United States

MARTA C GONZÁLEZ, University of California, Berkeley, Berkeley, CA, United States

Open Access Support provided by:

University of California, Berkeley

Published: 03 November 2025

[Citation in BibTeX format](#)

SIGSPATIAL '25: 33rd ACM International Conference on Advances in Geographic Information Systems
November 3 - 6, 2025
MN, Minneapolis, USA

Conference Sponsors:
SIGSPATIAL

sparkmobility: A Spark-based Python Library for Processing, Modeling, and Analyzing Large Mobility Datasets

Shangqing Cao*

Department of Civil and Environmental Engineering, UC
Berkeley
Berkeley, California, USA
caoalbert@berkeley.edu

Marta C. González (advisor)

Department of Civil and Environmental Engineering, UC
Berkeley
Berkeley, California, USA
martag@berkeley.edu

CCS Concepts

• **Information systems** → **Geographic information systems**;
Location based services.

Keywords

Human Mobility Dataset, Apache Spark

ACM Reference Format:

Shangqing Cao and Marta C. González (advisor). 2025. sparkmobility: A Spark-based Python Library for Processing, Modeling, and Analyzing Large Mobility Datasets. In *The 33rd ACM International Conference on Advances in Geographic Information Systems (SIGSPATIAL '25)*, November 3–6, 2025, Minneapolis, MN, USA. ACM, New York, NY, USA, 2 pages. <https://doi.org/10.1145/3748636.3766538>

1 Introduction

Location-Based Service (LBS) data, collected from personal mobile devices, have enabled significant advances in understanding human mobility patterns over the past decade. Extracting insights from these datasets typically involves using complex data-mining algorithms to detect, filter, and cluster stay locations. However, LBS datasets are often massive—ranging from tens to hundreds of gigabytes per day—posing serious computational challenges for traditional data processing tools. Libraries such as Pandas operate in a single-machine environment and require the entire dataset to fit into memory, making them unsuitable for processing LBS data at scale [3]. sparkmobility allows students and researchers to process large LBS dataset with improved memory management.

Since its release in 2012, Apache Spark has emerged as the de facto standard for big data analytics. Its distributed computing model allows large datasets to be processed in parallel across multiple machines. Key features like lazy evaluation, in-memory computation, and resilient distributed datasets (RDDs) enable Spark to overcome memory limitations and handle iterative and large-scale data transformations efficiently. In this paper, we introduce sparkmobility, a Spark-based framework designed to perform scalable processing and analysis of LBS data using these capabilities.

*ACM student member: 1057350



This work is licensed under a Creative Commons Attribution 4.0 International License. *SIGSPATIAL '25, Minneapolis, MN, USA*

© 2025 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-2086-4/2025/11

<https://doi.org/10.1145/3748636.3766538>

2 Software Architecture

Shown in Fig. 1, Sparkmobility is composed of two components: a Scala-based backend¹ and a Python front end². The data processing pipelines are implemented in Scala, Spark’s native language, and compiled into a Java Archive (JAR) file. The Python library serves as a lightweight interface to this backend, which allows the users to access the compiled pipelines without having to interact with the underlying Scala code or compiling locally. In addition to serving as a wrapper, the Python library also includes human mobility models that can directly consume the processed outputs within Uber’s H3 spatial framework.

Developers can contribute to either the Scala repository or the Python interface, while end-users primarily interact with the Python API. To streamline setup, the JAR file is hosted on Google Cloud Storage (GCS) and is automatically downloaded when the Python package is loaded in a Python environment for the first time. This eliminates the need for users to compile the pipelines locally. Furthermore, sparkmobility-py automatically retrieves and installs a compatible version of Apache Spark upon its first use.

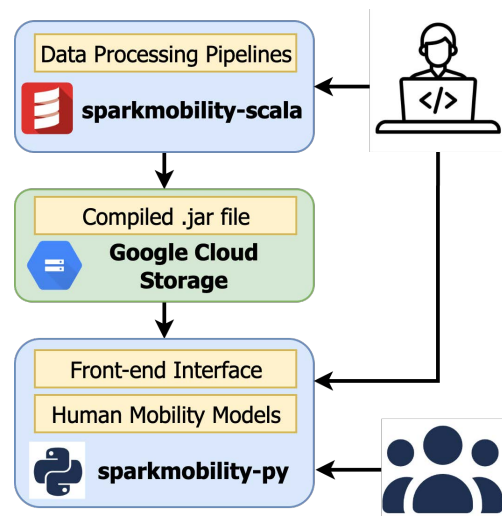


Figure 1: Overall software architecture. Developers interact with both the Scala and Python code space but users only interact with the Python code space.

¹<https://github.com/humnetlab/sparkmobility-scala>

²<https://github.com/humnetlab/sparkmobility-testing>

caid h3_region_stay_id stay_start_timestamp stay_end_timestamp stay_duration h3_id_region local_time h3_index	
0006938866763f742...	0 2019-01-01 01:38:50 2019-01-01 01:38:50 INTERVAL '0 00:00... 613221929790210047 2018-12-31 19:38:50 8829A189D9FFFFFF
0006938866763f742...	1 2019-01-01 02:49:06 2019-01-01 03:31:41 INTERVAL '0 00:42... 613221929314156543 2018-12-31 20:49:06 8829A18813FFFFFF
0006938866763f742...	2 2019-01-01 04:51:29 2019-01-01 04:51:29 INTERVAL '0 00:00... 613221929314156543 2018-12-31 22:51:29 8829A18813FFFFFF
0006938866763f742...	3 2019-01-01 05:58:02 2019-01-01 08:38:14 INTERVAL '0 02:40... 613221929314156543 2018-12-31 23:58:02 8829A18813FFFFFF
00185451363399505...	0 2019-01-01 07:43:07 2019-01-01 09:28:28 INTERVAL '0 01:45... 613220280510185471 2019-01-01 01:43:07 88298989CDFFFFF
00185451363399505...	1 2019-01-01 10:56:00 2019-01-01 11:24:56 INTERVAL '0 00:28... 613220280510185471 2019-01-01 04:56:00 88298989CDFFFFF

Figure 2: Sample output of the stay detection pipeline. The raw records are filtered and aggregated into stay points within each region defined by h3_index in the output.

2.1 Data Processing Pipelines

The data processing pipelines, implemented in Scala using the Spark framework, are designed for efficient and scalable processing of large-scale mobility datasets. A mobility data set is defined as a collection of spatial-temporal observations, where each record is a tuple $r = (i, t, x, y)$: i denotes the user identifier, t is the timestamp, and (x, y) represents the longitude and latitude coordinates. The primary objective of the pipeline is to extract stay points and infer the home and work locations of each user.

We adopt the stay detection method proposed by Zheng et al. [5]. Raw records are first filtered using spatial-temporal criteria: consecutive records within 5 minutes and 300 meters are grouped into a candidate stay point. The duration of a candidate stay is computed as the time span between the earliest and latest records in the group. A hierarchical mapping algorithm is then applied to aggregate candidate stay points into final stay points using a grid system at a specified resolution. To allow convenient indexing and processing of areas and locations, we use the hexagonal grid system defined in the H3 package instead of the square grid used in the original algorithm in [5]. Figure 2 shows an example output of the stay detection pipeline.

2.2 Human Mobility Models

Human mobility models are used to characterize the human mobility patterns that exist in the underlying human mobility datasets. These models often use static laws and machine learning tools to describe the movement of individuals and then generate realistic trajectories following the discovered patterns [3]. sparkmobility-py hosts the implementation of human mobility models that are implemented within the H3 framework. We include exploration and preferential return (EPR) [4], gravity model [1], and TimeGeo [2] in the release.

3 Experimental Results

To demonstrate the computational advantage of sparkmobility in processing large-scale human mobility data, we compare the execution time of the stay detection algorithm described in Section 2.1 with the C++ implementation by Jiang et al.[2]. The evaluation uses a test dataset provided by Veraset, covering approximately 2.5 million daily users in the US. As shown in Fig. 3A), sparkmobility achieves an order of magnitude reduction in computation time compared to the C++ baseline. For all but the smallest sample size (approximately 1% of users), the improvement made by Spark’s processing implementation outweighs the initial overhead cost. To validate the resulting stay points in sparkmobility, we examine

the distribution of the number of daily visited locations. Figure 3 B) shows that the result from sparkmobility is consistent with the C++ implementation.

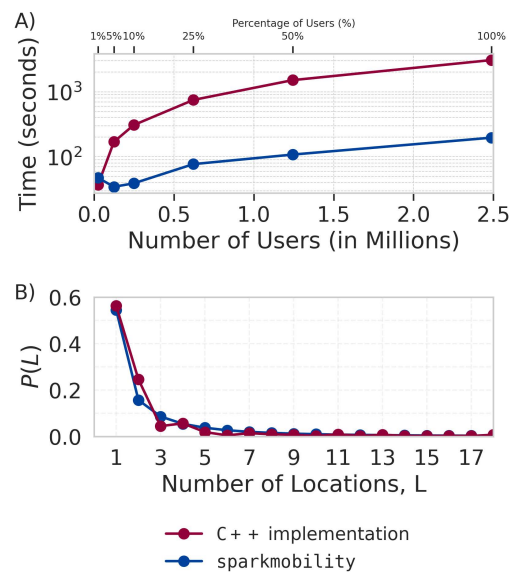


Figure 3: Experimental results. A) Comparison of execution time between the C++ implementation and sparkmobility. B) Distribution of the number of daily visited locations.

References

- [1] Hugo Barbosa, Marc Barthelemy, Gourab Ghoshal, Charlotte R. James, Maxime Lenormand, Thomas Louail, Ronaldo Menezes, José J. Ramasco, Filippo Simini, and Marcello Tomasini. 2018. Human mobility: Models and applications. *Physics Reports* 734 (March 2018), 1–74. doi:10.1016/j.physrep.2018.01.001
- [2] Shan Jiang, Yingxiang Yang, Siddharth Gupta, Daniele Veneziano, Shounak Athavale, and Marta C. González. 2016. The TimeGeo modeling framework for urban mobility without travel surveys. *Proceedings of the National Academy of Sciences* 113, 37 (Sept. 2016). doi:10.1073/pnas.1524261113
- [3] Luca Pappalardo, Filippo Simini, Gianni Barlacchi, and Roberto Pellungrini. 2022. **scikit-mobility**: A Python Library for the Analysis, Generation, and Risk Assessment of Mobility Data. *Journal of Statistical Software* 103, 4 (2022). doi:10.18637/jss.v103.i04 Publisher: Foundation for Open Access Statistic.
- [4] Chaoming Song, Tal Koren, Pu Wang, and Albert-László Barabási. 2010. Modelling the scaling properties of human mobility. *Nature Physics* 6, 10 (Oct. 2010), 818–823. doi:10.1038/nphys1760 Publisher: Springer Science and Business Media LLC.
- [5] Vincent W. Zheng, Yu Zheng, Xing Xie, and Qiang Yang. 2010. Collaborative location and activity recommendations with GPS history data. In *Proceedings of the 19th international conference on World wide web*. ACM, Raleigh North Carolina USA, 1029–1038. doi:10.1145/1772690.1772795