A Passenger Flow Analysis Method Through Ride Behaviors on Massive Smart Card Data

Weilong Ding[1,2(✉)], Zhuofeng Zhao[1,2], Han Li[1,2], Yaqi Cao[3], and Yang Xu[4]

[1] Data Engineering Institute, North China University of Technology, Beijing 100144, China
dingweilong@ncut.edu.cn
[2] Beijing Key Laboratory on Integration and Analysis of Large-Scale Stream Data, Beijing 100144, China
[3] Laboratory of Parallel Software and Computational Science, Institute of Software, Chinese Academy of Sciences, Beijing 100080, China
[4] Beijing E-Hualu Information Technology Co., Ltd., Beijing 100043, China

Abstract. In transportation business, the *passenger flow* analysis counts the ridership of given bus stops on given time duration. On the smart card data from the card readers of buses, the calculation of passenger flow faces challenges: the accuracy or the latency is blamed, and the scalability is poor on large volume data. In this paper, we propose an effective method on massive smart card data, in which ride behaviors are modeled and the passenger flow can be achieved and efficiently. Our method is implemented by Hadoop MapReduce, and proves minute-level latencies on weekly historical data with nearly linear scalability.

Keywords: Smart card data · Passenger flow · Ride behavior
Urban computing

1 Introduction

In urban transportation, the *passenger flow* analysis counts the ridership of given bus stops on given time duration. It has been widely applied to find hot spots, improve bus scheduling and evaluate service quality, and is essential to build industrial intelligent system [1]. Accordingly, related research is always hot in academia. Traditionally, smart card data of buses is stored in data warehouse or relational database, and the passenger flow analysis is done by SQL composition or store procedure [2, 3] through statistic model on small samples. However, it still faces inherent limitations on massive data. First, the latency is intolerable when large data involved. To achieve the latest values, the database suffers long time because huge volume data should be loaded, scanned and sorted several times during query execution. Second, the accuracy is blamed due to complex ride behaviors of passengers. With simplified assumptions, traditional methods on small samples [4] could only count short-time passenger flow (e.g., five minutes duration) at limited bus stops. Third, the scalability is extremely poor when data or infrastructure grows.

I. Romdhani et al. (Eds.): CollaborateCom 2017, LNICST 252, pp. 374–382, 2018.
https://doi.org/10.1007/978-3-030-00916-8_35

In this paper, we propose a novel analysis method for different types of passenger flow through ride behavior models. The contributions can be summarized as follows. (1) Ride behavior is modeled by business rules and statistics, and passenger flow would be achieved accordingly in an efficient way. (2) Passenger flow depicted as MapReduce jobs proves minute-level latencies on weekly data with nearly linear scalability in extensive conditions.

2 Motivation and Related Work

Our work was initiated by *Passenger Big Data Analysis Platform* in Beijing. We collaborated with *E-hualu*, one of the leader companies in China for intelligent transportation system (ITS). We had deployed a bus scheduling system for more than 30 new night-bus lines in late 2014, and were eager to improve the bus departure intervals and the passenger on-board time through Big Data technologies. A record as the data unit contains 13 attributes in Table 1 including three entities, two timestamps and two spatial attribute-groups. Currently, the smart card data was regularly processed through traditional databases, while the execution latency on massive data is too long to endure. It is urgent to find effective solutions, and that is our original motivation.

Table 1. A record structure of smart card data.

Attribute	Notation	Type
card_ID	Identity of smart card	Entity
line_ID	Identity of bus line	
bus_ID	Identity of bus	
begin_time	Timestamp of getting-on	Time
end_time	Timestamp of getting-off	
from_station_ID	Identity of getting-on station	Space
from_station_name	Name of getting-on station	
from_station_longitude	Longitude of getting-on station	
from_station_latitude	Latitude of getting-on station	
to_station_ID	Identity of getting-off station	
to_station_name	Name of getting-off station	
to_station_longitude	Longitude of getting-off station	
to_station_latitude	Latitude of getting-off station	

There is an assumption that the data has been cleaned to eliminate fallacious records. The low data quality obstructs the data analysis and user experience [5]. We employ data cleaning method previously proposed [6] on those massive spatio-temporal data to guarantee temporal consistency and semantic legality.

Database is the traditional technology for passenger flow analysis. Through persistent data storage, the smart card data is processed by SQL composition or store procedure [2]. On the card data in GTFS (General Transit Feed Specification) format, Tao et al. [7] demonstrate a multi-step methodology to examine the spatial-temporal

dynamics of travel behaviors among bus passengers. But such technology suffers long latency, because holistic data ought to be scanned several times during execution [8].

To improve the latency, the statistic model is widely used on small samples to predict passenger flow in short-time. Ma et al. [2] build travel probability models on smart card data, and their DBSCAN joint algorithm could identify historical travel patterns and regularities. Zhang et al. [4] propose a Kalman filter method to forecast short-term passenger flow on heterogeneous data (including smart card data of buses). Their accuracy is high only on limited data, but cannot hold on large volume.

Big data technology is being adopted because traditional models cannot run well in continuous and scalable environment [9]. Through Hadoop, the intelligent transportation systems like SMARTBUS [10] and Xiong's work [1] integrate data in multiple layers and views. Through MapReduce on smart card data and bus GPS data, Zhang et al. [11] analyzed the passenger density to infer crowdedness and evaluate the vehicle scheduling, and Wang et al. [12] estimated boarding stop time and bus arrival time. All those works show their efficiency in specific business, but none of them has reported the passenger flow analysis yet, and we have to learn and start-up elaborately.

In brief, on massive smart card data of buses, it still lacks effective methods to analyze passenger flow.

3 Passenger Flow Analysis Through Ride Behaviors

3.1 Methodology

To take a bus at one stop, a passenger would have one of three behaviors: getting-on, getting-off and transfer. In this paper, we focus on the getting-on behavior and its passenger flow.

Definition 1. Getting-on (getting-off) passenger flow. For a bus stop s on any time duration δ_l by interval θ, the passenger flow of getting-on (getting-off) is the count of passengers who gets on (off) any bus b_i at s. $|l|$ = a day/θ, $l > 0$.

The time durations are divided by the time *interval* θ which can be assigned as 30 min, 1 h or 1 day in practice. Therefore, the *passenger flow analysis* in this paper is the calculation to achieve the passenger flow defined above at all the bus stops on any duration. The getting-on and getting-off analyses are similar due to their symmetrical behaviors, so that only the former one would be discussed in details in the following parts.

The architecture of passenger flow includes three layers as a PaaS (platform as a service) fashion.

The bottom is data layer to maintain two kinds of data. The basic data is the dependent items for analysis, such as bus stop, bus line and vehicle id, which is preserved in relational database like MySQL. The smart card data would be uploaded daily from offline files to distributed file system HDFS after data cleaning. Compared with the basic data, the smart card data has much larger volume (1 TB vs. 100 MB) and updates in higher frequency (daily vs. weekly).

The middle is processing layer for analysis execution. Different types of passenger flow would be completed as respective MapReduce jobs. The time interval θ is 1 h by

default, and could be adjusted on demand. Those jobs would be dispatched as parallel tasks. In fact, as an open layer, other jobs on those data rather than passenger flow analyses can be calculated here.

The top is the application layer to monitor jobs and get results. The *calculation console* is a web monitor for machines, services (e.g., jobtracker, tasktracker, namenode, datanode, zookeeper), and jobs (states, slot, task counter, load-balance). Some native commands like job submission and halt have been encapsulated as GUI (graphical user interface). The *map service* shows results in an online map, in which the bus stops, bus lines and vehicle trajectories are directly visualized. At selected bus stop, the analysis results can achieve intuitively in a pop-up table. The visual parameters like resolution and zoom can be also adjusted.

3.2 Getting-on Behavior and Its Passenger Flow

As the Definition 1, getting-on passenger flow ought to refer a location (i.e., bus stop) on given time (i.e., time durations).

There are three problems about time here. (1) One problem is to how discriminate a bus' different trips, when the records of the smart card data on the same bus were appended in sequential. A bus has high possibility to drive for service more than one round-trip in a day, and those different trips cannot be distinguished from each other directly. (2) The second problem is how to infer the stop and start time of a bus at one bus stop. The getting-on/off timestamp of passengers varies much due to the card tapping time of passengers. (3) The third problem is how to sensibly count passenger flow if the wait period $[t_m, t_n)$ of a bus overlaps two adjacent durations. For example, when θ = 1 h, t_m = 8:58, t_n = 9:01; which is more rational reflections about the ridership during the period $[t_m, t_n)$: the duration δ_8 = [7:00–8:00) or δ_9 = [8:00–9:00)?

To solve those problems, we observe the characteristics of data, and propose the following model with the symbols in Table 1.

Definition 2. Getting-on behavior model of passengers. For a bus stop s of bus b_j, the getting-on behavior in any record r conforms these characteristics:

(1) entity attribute $r.bus\_ID = b_j$;
(2) four spatial attributes of $r.from\_station\_*$ is related with bus stop s;
(3) all the long integers of temporal attribute $r.begin\_time$
   (a) can be clustered according to trips of bus service;
   (b) are smaller than the start time of b_j in high possibility;
   (c) are negative skewed because their mean is larger than their SD (standard deviation).

Here, the third one just reflects the facts of three problems above, and we would cope with them individually.

(a) To speed the clustering on those long integers to discriminate different trips, we include an auxiliary condition: for bus b_j, the successive records r_i and r_{i+1} belong to serial trips respectively if $(r_{i+1}.begin\_time - r_i.begin\_time) < \gamma$. We employ the empirical γ = 420 (i.e., 7 min), because we have learned from the official documents, a bus in Beijing spends at least 14 min for a round-trip.

(b) To infer the start time of bus b_j at stop s, we have to get rid of the skewness of original data. For the data in skewed distribution, logarithm transformation is feasible to find the normality, because it can turn multiplicative effects into additive ones. The calculation is the following procedure:
   i. for any $r_i.begin\_time$, transform it to its Natural Logarithm value g_i;
   ii. on all those logarithm values, calculate their mean m_g and SD sd_g;
   iii. the start time can be inferred as $t_m = \text{EXP}\ (m_g + 2^*\ sd_g)$, where EXP $(x) = e^x$.

(c) To count passenger flow in a time period overlapped two adjacent durations δ_i, δ_{i+1}, we refer the start time t_m got from (b): the getting-on ridership belongs to either δ_i if $t_m \in \delta_i$ orδ_{i+1} (i.e., $t_m \in \delta_{i+1}$). It is sound because the getting-on behavior depends on action of bus start.

Through the behavior model in Definition 2, the getting-on passenger flow analysis can be designed as two-step procedure in Fig. 1, and each step is implemented as a Hadoop MapReduce job. In this figure, the left part of either step is realized as a map task and the right one is a reduce task; each of which requires only one-pass to scan the data.

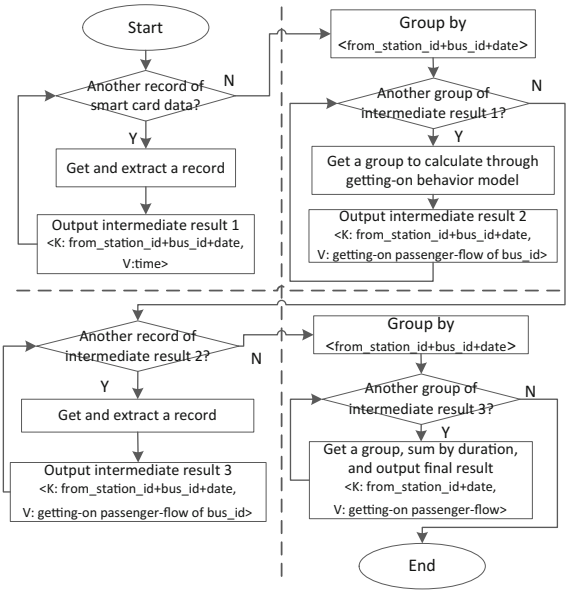

Fig. 1. Getting-on passenger flow analysis

The first step is to achieve the getting-on passenger flow of single bus. It is represented in the top part of the Fig. 1. Here, each record would be extracted by its attributes. The timestamp of getting-on (i.e., attribute *begin_time*) is divided by two parts: *date* and *time*. After grouping by the composition of station id, bus id and date, the calculation is done through the getting-on behavior model. The intermediate results

could be sorted by time and then be outputted as the getting-on passenger flow of each bus. For example, if θ is set as 1 h, a output could be <3, 00028294, 20151208, 0, 0, 0, 0, 0, 0, 13, 0,0, 25, 0, 0, 0, 18, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0>. It means the bus 0028294 at bus stop 3 on Dec. 8th 2015 has three trips when the ridership is 13 in [6:00, 7:00), 25 in [9:00, 10:00) and 18 in [13:00, 14:00).

The second step is to achieve the getting-on passenger flow of all the buses. It is expressed as the bottom part of Fig. 1. In this step, the output of the first step is read, and each record would be extracted by *from_station_id, bus_id, date* and vectors of *passenger flow* of single bus. After grouping by the composition of station id, bus id and date, the intermediate results could be sorted by time and then be outputted as the final getting-on passenger flow. For example, if θ is set as 1 h, a output could be <3, 20151208, 0, 0, 0, 0, 0, 64, 85, 105,128, 256, 204, 230, 242, 189, 205, 143, 145, 252, 286, 259, 235, 102, 82, 35>. It shows the getting-on passenger flow at bus stop 3 on any duration of Dec. 8th 2015 divided by one hour interval.

The **getting-off behavior model** for **the getting-off passenger flow analysis** could be defined analogously. In that model, the different attribute $r_i.end\_time$ and the bus stop time are focused. The inferred stop time $t_n = \text{EXP}(m_g - 2^* sd_g)$ on logarithm values of getting-off attribute, because those timestamps are positive skewed and larger than stop time in high possibility.

4 Evaluations

4.1 Settings

The performance and effects are respectively evaluated as experiments and case studies in this section. Here, six virtual machines are used in our private Cloud to implement our method, each of which owns 4 cores CPU, 4 GB RAM and 1.2 TB storage with CentOS 6.6 x86_64 installed. Four Acer AR580 F2 rack servers via Citrix XenServer 6.2 were used for the virtualization, each of which own 8 processors (Intel Xeon E5-4607 2.20 GHz), 48 GB RAM and 80 TB storage.

We employ the smart card data of Beijing buses on eight days in 2013 which contains 24263142 records on 7349 buses of 233 lines involving 3581 bus stops. All the data was generated from the readers charging by distance, and each record contains 13 attributes as Table 1 in Sect. 2. The data has been cleaned in advance by our dedicated method [6]. To design the experiments on data in different volume, we have divided those 8-day data into eight parts by their dates.

Two different ways for the passenger flow analysis have been implemented for comparison. One is our method termed as BD (Big Data), in which passenger flow analysis are implemented as Hadoop MapReduce jobs. The counterpart is a statistical estimation method [13] termed as ODE (Origin-destination Estimation) in the current production environment.

4.2 Experiments

We compare and evaluate the performance through experiments below.

Experiment 1. The data of eight dates is used as the inputs of different volume. Run the two types of passenger flow analysis through both BD and ODE methods, and note their average executive times.

The result is showed as Fig. 2.

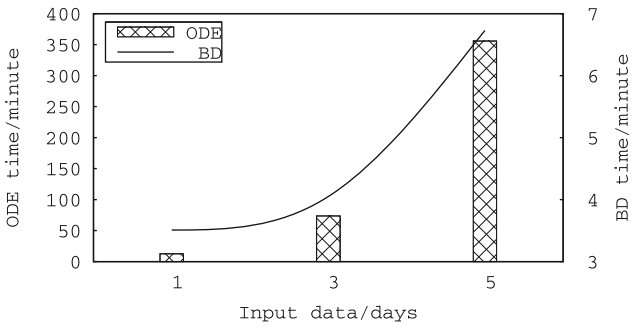

Fig. 2. The comparison of passenger flow analysis in different ways

As the input volume increases, the executive time through both methods rises, but through ODE it is longer than that of BD by two orders of magnitude in either analysis. The volume of three inputs is approximate 3 million, 5 million and 15 million respectively. The executive time through ODE grows sharply when the volume is bigger than 5 million, while that of BD rises almost linearly. On 3-day input, BD only consumes minute-level executive time, while the ODE requires more than 5 h. The low latency through BD comes from the parallel execution of two-step procedure for either analysis. But through ODE, the analysis requires multiple data scans to sort and query, and moreover it has to run on a single machine without parallelism.

We found the too long latencies through ODE, and only BD method is considered in the following experiments to evaluate the efficiency and scalability.

Experiment 2. The data of one day is appended to the input each test, and the executive times for getting-on analysis through BD are noted. For comparison, on each data volume, three different intervals, 10-min, 1-h and 4-h, are set respectively. The result is presented in Fig. 3(a). The average executive time on one million records in each test can be deduced as Fig. 3(b).

The getting-on analysis through BD method is proved scalable on data volumes. On the one hand, when input scales, the increment of executive time of getting-on analysis is better than linearity at any interval. In Fig. 3(a), the time is kept minute-level and not doubled even when the input grows eight folds. That trend can also be comprehended clearly from Fig. 3(b), where the average executive time on give size data declines to the steadiness about 10 s. It demonstrates that the processing capacity of BD method is stable and horizontally scalable. On the other hand, on the input of the same volume,

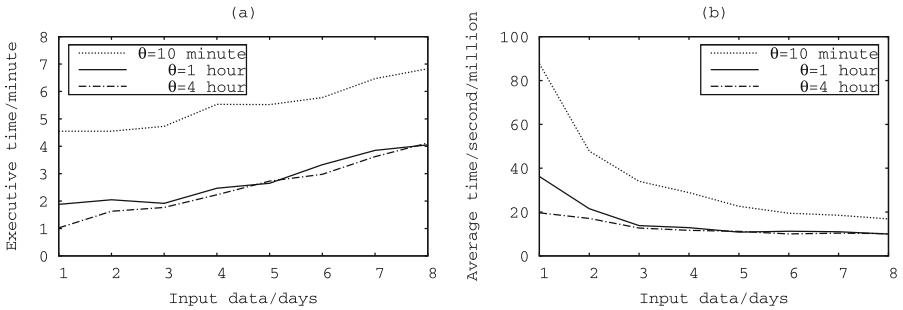

Fig. 3. The analysis under different intervals and inputs

the executive time varies by intervals. The larger interval has lower latencies, because either behavior model of BD method relies on the interval: smaller interval implies more time durations for calculation and requires more time to complete. It is interesting that when input scales, the capacity on given size data converges in any interval as Fig. 3(b), which also proves its scalability.

With the experiments above, our method proves minute-level latency on weekly data in a scalable fashion.

5 Conclusion

Through ride behavior model, we propose a method to analyze the passenger flow on massive smart card data. For any analysis on weekly historical data, our method can hold minute-level latency and keep nearly linear scalability in extensive conditions. It also efficiently shows practical results efficiently. Furthermore, we would research other rider behaviors, and would analyze their passenger flow through respective behavior models.

Acknowledgments. This work was supported by the Youth Program of National Natural Science Foundation of China (Nos. 61702014, 61602437), Beijing Natural Science Foundation (No. 4162021), R&D General Program of Beijing Education Commission (No. KM2015_1000 9007), the Key Young Scholars Foundation for the Excellent Talents of Beijing (No. 20140000 20124G011), Top Young Innovative Talents of North China University of Technology (Nos. XN018022, XN072-001) and Youth Innovation Foundation of North China University of Technology (No. 1473-1743028).

References

1. Xiong, G., Zhu, F., Dong, X., Fan, H., Hu, B., Kong, Q., Kang, W., Teng, T.: A kind of novel ITS based on space-air-ground big-data. IEEE Intell. Transp. Syst. Magn. **8**, 10–22 (2016)
2. Ma, X., Wu, Y.-J., Wang, Y., Chen, F., Liu, J.: Mining smart card data for transit riders' travel patterns. Transp. Res. C Emerg. Technol. **36**, 1–12 (2013)

3. Long, Y., Zhang, Y., Cui, C.: Identifying commuting pattern of Beijing using bus smart card data. Acta Geogr. Sin. **67**, 1339–1352 (2012). (in Chinese)
4. Chunhui, Z., Rui, S., Yang, S.: Kalman filter-based short-term passenger flow forecasting on bus stop. J. Transp. Syst. Eng. Inf. Technol. **11**, 154–159 (2011). (in Chinese)
5. Carey, M.J., Jacobs, S., Tsotras, V.J.: Breaking BAD: a data serving vision for big active data. In: Proceedings of the 10th ACM International Conference on Distributed and Event-Based Systems, pp. 181–186. ACM, Irvine (2016)
6. Ding, W., Cao, Y.: A data cleaning method on massive spatio-temporal data. In: Wang, G., Han, Y., Martínez Pérez, G. (eds.) APSCC 2016. LNCS, vol. 10065, pp. 173–182. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-49178-3_13
7. Tao, S., Rohde, D., Corcoran, J.: Examining the spatial–temporal dynamics of bus passenger travel behaviour using smart card data and the flow-comap. J. Transp. Geogr. **41**, 21–36 (2014)
8. Dugane, R.A., Raut, A.: A survey on big data in real time. Int. J. Recent Innov. Trends Comput. Commun. **2**, 794–797 (2014)
9. Pelletier, M.-P., Trépanier, M., Morency, C.: Smart card data use in public transit: a literature review. Transp. Res. C Emerg. Technol. **19**, 557–568 (2011)
10. Ram, S., Wang, Y., Currim, F., Dong, F., Dantas, E., Saboia, L.A.: SMARTBUS: a web application for smart urban mobility and transportation. In: Proceedings of the 25th International Conference Companion on World Wide Web, pp. 363–368. International World Wide Web Conferences Steering Committee, Montreal (2016)
11. Zhang, J., Yu, X., Tian, C., Zhang, F., Tu, L., Xu, C.: Analyzing passenger density for public bus: inference of crowdedness and evaluation of scheduling choices. In: 17th International IEEE Conference on Intelligent Transportation Systems (ITSC 2014), pp. 2015–2022. IEEE (2014)
12. Wang, Y., Ram, S., Currim, F., Dantas, E., Saboia, L.A.: A big data approach for smart transportation management on bus network. In: 2016 IEEE International Smart Cities Conference (ISC2), pp. 1–6. IEEE (2016)
13. Xuemei, Z., Xiyu, Y., Xiaofei, W.: Origin-destination matrix estimation method of public transportation flow based on data from bus integrated-circuit cards. J. TongJi Univ. (Nat. Sci.), 1027–1030 (2012). (in Chinese)