Data Science and Engineering (2018) 3:52–67
https://doi.org/10.1007/s41019-018-0061-9

Bus-OLAP: A Data Management Model for Non-on-Time Events Query Over Bus Journey Data

Lei Duan[1] · Tinghai Pang[1] · Jyrki Nummenmaa[2,3] · Jie Zuo[1] · Peng Zhang[1] · Changjie Tang[1]

Received: 6 September 2017 / Revised: 26 October 2017 / Accepted: 28 February 2018 / Published online: 15 March 2018
© The Author(s) 2018

Abstract
Increasing the on-time rate of bus service can prompt the people's willingness to travel by bus, which is an effective measure to mitigate the city traffic congestion. Performing queries on the bus arrival can be used to identify and analyze various kinds of non-on-time events that happened during the bus journey, which is helpful for detecting the factors of delaying events, and providing decision support for optimizing the bus schedules. We propose a data management model, called Bus-OLAP, for querying bus journey data, considering the characteristics of bus running and the scenarios of non-on-time analysis. While fulfilling typical requirements of bus journey data queries, Bus-OLAP not only provides a flexible way to manage the data and to implement multiple granularity data query and update, but it also supports distributed queries and computation. The experiments on real-world bus journey data verify that Bus-OLAP is effective and efficient.

Keywords Data management · OLAP · Parallel computing

1 Introduction

Public bus service plays an important and irreplaceable role in the traffic system of a city. On one hand, public bus is one of the most convenient and cost-efficient ways for people to travel. On the other hand, public bus service, as an alternative to the use of private cars, is an effective way to reduce carbon dioxide emissions. Promoting the bus service not only provides convenience for people but it also improves the urban living conditions and helps in the fight against the climate change. *On-time bus* is an emerging bus running mode where the bus arrives at each bus stop according to the time table strictly, which is helpful for passengers to avoid wasting too much time at bus stops. There are several countries, e.g., the USA, Finland, and Japan, where efforts have been made to implement on-time bus running mode to improve the bus service quality.

Clearly, a reasonable design of bus running routes and time tables is the key to carry out the on-time bus running mode. To initiate this development, practical bus journey data including the arrival information of a bus at every stop should be collected at first. The developments of sensors and Internet of Things enable the automatic collection of bus journey data. For example, the local government of Tampere, Finland, started in 2015 to record and publish as open data the location information of each running bus

This work was supported in part by the National Natural Science Foundation of China under grant No. 61572332, the Fundamental Research Funds for the Central Universities under grant No. 2016SCU04A22, the China Postdoctoral Science Foundation under grant No. 2016T90850, and the Academy of Finland Foundation under grant No. 295694.

✉ Lei Duan
leiduan@scu.edu.cn

Tinghai Pang
pangtinghai@163.com

Jyrki Nummenmaa
Jyrki.Nummenmaa@staff.uta.fi

Jie Zuo
zuojie@scu.edu.cn

Peng Zhang
zp_jy1993@163.com

Changjie Tang
cjtang@scu.edu.cn

[1] School of Computer Science, Sichuan University, Chengdu, Sichuan, China

[2] Faculty of Natural Sciences, University of Tampere, Tampere, Finland

[3] Sino-Finnish Centre, Tongji University, Shanghai, China

Table 1 Instances of bus journey record

Stop	Name	Lat.	Long.	Line	Date	Arrival	Delay
1500	Ammattikoulu	61.4986	23.7355	13	2016-08-03	16:40:15	210
3084	Kuoppamaentie	61.4799	23.8046	560	2016-08-04	17:20:09	610
3084	Kuoppamaentie	61.4799	23.8046	560	2016-08-05	00:09:48	− 34
0098	Savilinna	61.5000	23.7374	28	2016-08-06	16:55:12	340

real-time every second. Particularly, the essential bus journey data include information on, for each running bus at each bus stop location, whether the bus was on schedule at a stop and how long was the delay if it was not on time. Table 1 lists several samples of Tampere bus journey data. There, each record contains the information of a bus arriving at a bus stop.

Example 1 In Table 1, "Stop" represents the bus stop ID, "Name" is the name of the bus stop, "Latitude" and "Longitude" indicate the location of a bus stop, "Line" is the bus line number, "Date" and "Arrival" are the date and time of bus arrival, respectively, and "Delay" is the difference between the arrival time and the scheduled time ('−' indicates that bus arrives ahead of scheduled time). For instance, from the first record in Table 1, we can see that a bus on Line 13 arrived at Stop Ammattikoulu (ID: 1500, location: 61.4986″N, 23.7355″E) at 16:40:15 PM on August 3, 2016, being 210 s behind the schedule.

A *bus non-on-time query* provides statistics on the non-on-time arrivals of buses that happened under given conditions. Bus non-on-time queries are useful for the bus route design, online information systems on bus transportation, and the bus running schedule optimization. In general, the non-on-time events can be divided into the early case (the arrival time is ahead of the scheduled time) and the delaying case (the arrival time is later than the scheduled time). Please note that the main concern of bus non-on-time query is the delaying case, since (i) the early case can be easily avoided by slowing or stopping buses in the real-life situation, and (ii) the techniques supporting the query for delaying cases can be used for early cases. Thus, for the sake of simplicity, we just discuss the delaying case hereafter. However, the results generalize to the early case.

Typical bus non-on-time queries [15, 18] include: "How many bus delaying events happened over a given time period?", "Which are the routes where most delaying events happen?", and "Do the bus delaying events aggregate and where?", etc. Based on the query conditions, there are three kinds of bus non-on-time queries:

– Temporal queries, in which the query conditions are related to bus running time. For example, how many buses were delayed during $17:00 \sim 19:00$?
– Spatial queries, in which the query conditions are related to bus stop locations. For example, given a bus stop ID "560", which are the nearest neighboring bus stops where delaying events happen?
– Spatial–temporal queries, in which the query conditions are related to both bus running time and bus stop locations. For example, which are the spatial–temporal zones with significant aggregation of delaying events?

To the best of our knowledge, there is no data management model dealing with the bus non-on-time queries.[1]

Since non-on-time query over bus journey data can be used to improve bus service, thus improving comfortable travel experience, we should provide an efficient model to manage bus journey data. However, there are some challenges that we need to address: (i) How to manage the bus journey data? (ii) How to fuse data from different sources? (iii) How to perform queries efficiently? In this paper, to tackle these challenges related to non-on-time queries, we propose a model named Bus-OLAP to manage bus journey data.

In this paper, we make the following concrete contributions: (i) We build indexes of bus journey data by bit-vectors according to the application scenarios, taking into account particularly the temporal and spatial factors. (ii) We introduce efficient index operations to support the non-on-time queries on bus journey data. (iii) We implement the related computations on Spark to support distributed computing for the processing of the queries, thus enabling real-time response for the analysis for massive datasets.

The rest of the paper is organized as follows. We review the related work in Sect. 2 and present the critical techniques of Bus-OLAP in Sect. 3. In Sect. 4, we report a systematic empirical study using real-world bus data and we conclude the paper in Sect. 5.

2 Related Work

Our study is related to previous studies on urban traffic analysis, spatial–temporal data management, and parallel computing.

[1] We tackled the problem of bus non-on-time query in Pang et al [11], a preliminary version of this paper. The difference between these two works is presented in related works. We will review the related works in Sect. 2.

2.1 Urban Traffic Analysis

The analysis on urban traffic plays an important role and attracts extensive attention from public [25]. Traffic analysis helps to alleviate urban traffic congestion thereby improving environmentally friendly traffic for people to travel. Traffic analysis has been recently under active research. Yuan et al. [22] analyze the movements of taxis and passengers and provide an easy way for passengers to pick taxis. Pang et al. [10] detect the anomalous behavior of taxis in Beijing metropolitan area. Chen et al. [2] propose a model given the past trajectories and predict the next station of an object. Kong et al. [8] aim to deal with the problem of traffic congestion and propose a method to predict the traffic congestion using the floating car trajectories data. Han and Moutarde [6] focus on traffic dynamic prediction in urban transportation network and find out the evolution of traffic states, which contributes to the adjustment of traffic management. Zhang et al. [24] apply deep learning to predict the traffic of crowds in different regions of a city. Bao et al. [1] propose a data-driven approach to develop bike lane construction plans satisfying the constraints and objectives requested by the user. There are several works on predicting the urban traffic flow by the traffic state [4, 13, 17]. Wu et al. [18] detect the temporal–spatial aggregation of bus delay but do not consider the management of bus data.

To the best of our knowledge, the existing work about traffic analysis concentrates on the problems with vehicles with stochastic trajectories. However, the analysis on bus data is essentially different from on other vehicles. On one hand, the trajectory and schedule of each bus are fixed. On the other hand, the main concern on bus journey data is non-on-time analysis. Clearly, the existing methods proposed by previous work on traffic analysis are not applicable to online bus journey data analysis, and, in particular, they do not give an efficient method to manage large bus data sets for efficient online analysis.

We tackled the problem of bus journey data management in Pang et al. [11], a preliminary version of this paper. In comparison with that work, we have now extended the paper in the following aspects: (i) adding more introduction to the background of the research on bus journey data collection and analysis; (ii) including the necessary background knowledge about bus journey data preprocessing in Sect. 3.1; (iii) presenting a method of querying data from different sources; (iv) providing a more detailed description of key techniques for bus non-on-time query in Bus-OLAP; and (v) performing more extensive empirical evaluations.

2.2 Spatial–Temporal Data Management

Urban traffic data have spatial–temporal characteristics. The bus data management can improve the efficiency of storage, indexing, and querying of the data. Sistla et al. [12] propose a model, called MOST, to model the moving object with a time-related function, which improves the efficiency of storage and querying of moving objects in a database. Ting et al. [16] present a simplistic network model for moving objects. Some index structures such as R-tree [5] and B^{+}-tree [7] are also used to optimize spatial–temporal queries. Yu et al. [21] propose an algorithm, named YPK-CNN, to monitor KNN queries.

The works discussed above can improve the query efficiency by different ways. However, we want to provide a model by which all bus running related factors can be considered. For example, besides the factors listed in Table 1, some external factors, such as weather activity and temperature, also affect the on-time rate of bus service. In particular, we need a data management model that is suitable for a distributed computing environment.

2.3 Parallel Computing

Considering the requirement of analyzing large-scale bus journey data, parallel computing techniques are necessary to speed up the efficiency of spatial–temporal analysis [9]. A lot of work has been done on adapting parallel computing methods to support efficient queries over large-scale data. Xia et al. [19] design a method to conduct KNN queries based on Map-Reduce and apply it to real-time prediction of traffic flow. Eldawy et al. [3] have built a system, named GeoSpark, based on Spark to improve the efficiency of spatial–temporal data analysis. Xie et al. [20] apply R-tree indexing to parallel queries based on Spark, to deal with the efficiency issues of large-scale data.

In this work, we use Spark to build a parallel processing model based on bit-vector operations to conduct bus non-on-time queries.

3 Design of Bus-OLAP

In this section, we present our Bus-OLAP data management model for bus non-on-time queries. The framework of Bus-OLAP (Fig. 1) consists of data cleaning, transforming, loading, and querying. The preprocessing of bus journey data has been well studied in [14]. However, for the sake of self-completeness of our presentation, we briefly present the key points for data preprocessing here. Then, we discuss the most important techniques used in Bus-OLAP:

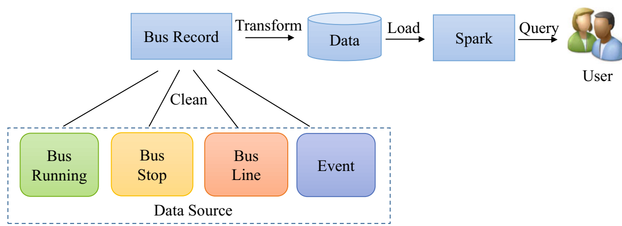


Fig. 1 Framework of Bus-OLAP

(i) data indexing, (ii) index operations, and (iii) parallel query computation.

3.1 Data Preprocessing

The raw bus journey data suffer from noise, inconsistencies, and missing observations [15]. This is due to the real-world situations, such as the location measurement inaccuracies of the global navigation satellite system, imprecise knowledge of the real observation time, and errors in data identification. Several methods had been used to clean the data. The noisy data can be identified as outliers by comparing with other data. The inconsistent data can be discarded with the help of known scheduled bus stop sequences. The missing data can be ignored, since there is typically enough data available for the analysis purposes.

Millions of observations are stored every day. However, the observations are not useful as such for statistical analysis. As a result, they are grouped into journeys, so that the essential bus running information, e.g., the bus route and the arrival time at each bus stop during one journey, can be studied. Furthermore, all bus journeys are segmented into links between subsequent bus stops, since the bus stop sequences and the location of each bus stop are available, and the segments between every two bus stops are useful for bus traffic analysis.

Example 2 In Fig. 2, there are two buses v_1 and v_2, three bus stops and four observations. Bus v_1 arrived at the bus stop "Rautatieasema" at 12:00 and then arrived at "Tammelan puistokatu" 3 min later. Thus, we grouped the first and third observation together because they are in the same journey of bus v_1. Similarly, the second and last observations in Fig. 2 belong to a journey of bus v_2. Furthermore, the link between bus stops "Rautatieasema" and "Tammelan puistokatu" is a segment of v_1's journey.

3.2 Data Indexing

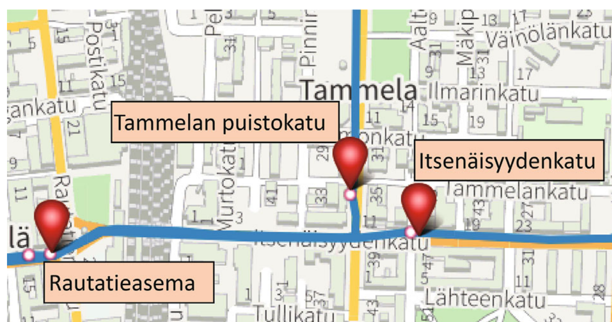
To support efficient bus non-on-time queries, we implement an index based on attribute domain partition in Bus-OLAP. Specifically, we divide the domain of each attribute into several disjoint partitions. Then, for each attribute value of a record, we build a bit-vector to indicate the partition it belongs to. Thus, we can apply bit operations to perform bus non-on-time queries, which improves the query efficiency.

Next, we introduce the details of the index building. For attribute A, the *domain* of A, denoted by $\mathcal{D}(A)$, is the set of all available values on attribute A. A *partition* of $\mathcal{D}(A)$, denoted by $\mathcal{P}_{\mathcal{D}}(A)$, is a collection of non-empty subsets of $\mathcal{D}(A)$ such that (i) $\mathcal{D}(A) = \bigcup_{d_i \in \mathcal{P}_{\mathcal{D}}(A)} d_i$, and (ii) for $\forall d_i, d_j \in \mathcal{P}_{\mathcal{D}}(A), i \neq j, d_i \cap d_j = \emptyset$. Please note that, in Bus-OLAP, the order of elements in $\mathcal{P}_{\mathcal{D}}(A)$ is predetermined and fixed during performing queries. We denote by $\mathcal{P}_{\mathcal{D}}(A)_{[i]}$ the ith element in $\mathcal{P}_{\mathcal{D}}(A)$. Furthermore, there may be several different partitions for an attribute. For example, the partitioning of $\mathcal{D}(``\text{Date}'')$ can be by month or by season.

For a record r, we denote by $r.A$ the value of r on attribute A. Given a partition $\mathcal{P}_{\mathcal{D}}(A)$ on attribute A, the index of $r.A$ is a *bit-vector* $< b_1, b_2, \ldots, b_{|\mathcal{P}_{\mathcal{D}}(A)|} >$ satisfying

$$b_i = \begin{cases} 1, & r.A \in \mathcal{P}_{\mathcal{D}}(A)_{[i]} \\ 0, & r.A \notin \mathcal{P}_{\mathcal{D}}(A)_{[i]} \end{cases} \tag{1}$$

Example 3 A partition on attribute "Date" according to day of the week is illustrated in Fig. 3. The domain of "Date" is partitioned into seven subdomains,



List of bus observations

ID	Bus	Arrival	Stop
1	v_1	12:00	Rautatieasema
2	v_2	12:01	Rautatieasema
3	v_1	12:03	Tammelan puistokatu
4	v_2	12:04	Itsenäisyydenkatu

Fig. 2 An example of bus journey

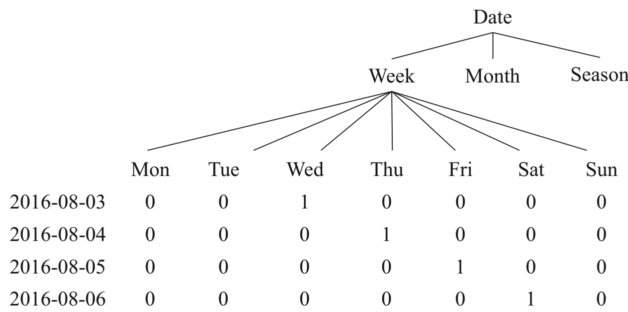


Fig. 3 An example of a partition on attribute "Date"

corresponding to the seven days of a week. The day "2016-08-03" in Table 1 represents August 3, 2016, and it is Wednesday. Therefore, the index of first record in Table 1 is $< 0, 0, 1, 0, 0, 0, 0 >$ under partition $\mathcal{P}_{\mathcal{D}}(\text{``Date''})$.

In Bus-OLAP, the selection of attribute partitions depends on:

- The query requirements. For example, we partition the domain of "Date" by day of the week and the domain of "Arrival" by time interval, so that we can find stops where most delaying events occur during the morning rush hour on Monday.
- The partition size. For example, we do not divide the domain of "Date" by date because the statistics on the non-on-time events within one day has little significance, and the space requirement following from the amount of subdomains (i.e., $|\mathcal{P}_{\mathcal{D}}(\text{``Date''})|$) would be large.

Table 2 lists the domain partition criteria for the main attributes listed in Table 1 by considering the application requirements.

For a partition $\mathcal{P}_{\mathcal{D}}(A)$ on attribute A, the index of all records is the set $V_A = \{v_1, v_2, \ldots, v_{|\mathcal{P}_{\mathcal{D}}(A)|}\}$, where $v_i \in V_A$ $(1 \le i \le n)$ is the column vector with respect to $\mathcal{P}_{\mathcal{D}}(A)_{[i]}$. If the value of the kth element of v_i is 1, the value of the kth record on attribute A belongs to $\mathcal{P}_{\mathcal{D}}(A)_{[i]}$.

Example 4 In Fig. 3, the column vector corresponding to "Wed" is $v = (1, 0, 0, 0)^{\mathrm{T}}$. The first element of v (i.e., 1) means that the value of the first record in Table 1 on attribute "Date" belongs to "Wed".

As discussed above, the non-on-time query over bus journey data can be easily implemented by vector operations. For a bitmap $V_A = \{v_1, v_2, \ldots, v_n\}$, we store $v_i \in V_A$ as a bit-vector. However, instead of loading all indexes, Bus-OLAP only loads the indexes (i.e., bit-vectors) of query-concerned attributes into memory. Then, a query can be converted into bit-vector operations.

For a new bus journey record, the index can be efficiently updated by appending a new bit at the end of each bit-vector.

3.3 Index Operations

For the sake of efficiency, we introduce some operations on indexes (bit-vectors) in Bus-OLAP. For a partition $\mathcal{P}_{\mathcal{D}}(A)$, we define the *subpartition* of $\mathcal{P}_{\mathcal{D}}(A)$ as $\hat{\mathcal{P}_{\mathcal{D}}}(A)$, where $\hat{\mathcal{P}_{\mathcal{D}}}(A) \subseteq \mathcal{P}_{\mathcal{D}}(A)$. The set of bit-vectors corresponding to $\hat{\mathcal{P}_{\mathcal{D}}}(A)$ is $\hat{V}_A = \{v_i \in V_A \mid \hat{\mathcal{P}_{\mathcal{D}}}(A)_{[i]} \in \mathcal{P}_{\mathcal{D}}(A)\}$. We define the OR operation on $\hat{V}_A$ as

$$\mathrm{OR}(\hat{V}_A) = v'_1 \mid v'_2 \mid \ldots \mid v'_m,$$

where $v'_i \in \hat{V}_A$ $(1 \le i \le m)$ and "|" denotes the bitwise OR operation between every two bit-vectors.

Clearly, the result of $\mathrm{OR}(\hat{V}_A)$ is also a bit-vector. The records satisfying subpartition $\hat{\mathcal{P}_{\mathcal{D}}}(A)$ can be obtained by index operations.

Example 5 Consider Fig. 3, and suppose the query target is finding the records generated in weekdays (from Monday to Friday). Firstly, we fetch the subpartition $\hat{\mathcal{P}_{\mathcal{D}}}(\text{``Date''}) = \{\text{Mon}, \text{Tue}, \text{Wed}, \text{Thu}, \text{Fri}\}$ from $\mathcal{P}_{\mathcal{D}}(\text{``Date''})$. As a result, the bit-vector set with respect to $\hat{\mathcal{P}_{\mathcal{D}}}(\text{``Date''})$ is $\hat{V}_{\text{``Date''}} = \{v_1, v_2, v_3, v_4, v_5\}$, where $v_1 = (0, 0, 0, 0)^{\mathrm{T}}$, $v_2 = (0, 0, 0, 0)^{\mathrm{T}}$, $v_3 = (1, 0, 0, 0)^{\mathrm{T}}$, $v_4 = (0, 1, 0, 0)^{\mathrm{T}}$, and $v_5 = (0, 0, 1, 0)^{\mathrm{T}}$. Then, $\mathrm{OR}(V_{\text{``Date''}}) = v_1 \mid v_2 \mid v_3 \mid v_4 \mid v_5 = (1, 1, 1, 0)^{\mathrm{T}}$. Thus, the records satisfying the query conditions are the first three records in Table 1.

In addition, we define the bitwise AND operation (denoted by&) for two bit-vectors belonging to different subpartitions, so that the query involving conditions on different attributes can be performed by index operations.

Example 6 Consider the example in Fig. 4. Suppose that the query target is the set of records whose "Longitude" ranges from 23.73 to 23.74, "Latitude" ranges from 61.48 to 61.50, and "Date" is Wednesday or Thursday. Then, the subpartitions of "Longitude," "Latitude," and "Date" are $\{o_1\}$, $\{a_3, a_4\}$, and $\{\text{Wed}, \text{Thu}\}$, respectively. The query result illustrated as the red area in Fig. 4 can be obtained by the following index operation: $\mathrm{OR}(\{a_3, a_4\}) \& \mathrm{OR}(\{o_1\}) \& \mathrm{OR}(\{\text{Wed}, \text{Thu}\})$.

3.4 Parallel Query Computation

Bus non-on-time query is computation intensive and response time sensitive. Figure 5 shows the framework of parallel query computation by bit-vectors using Spark [23].

Table 2 Attribute domain partitions in Bus-OLAP

Attribute	Criterion	Explanation
Latitude, Longitude	Distance	The geographic space of Tampere city is divided into a grid of 2000 × 1000 equal-sized rectangles. Each bus stop location is associated with a unique rectangle in the grid
Date	Week, Month, Season	The date granularity concerned by bus non-on-time query includes the day of the week ({Mon, Tue, ..., Sun}), month ({Jan, Feb, ..., Dec}), and season ({Spring, Autumn, Fall, Winter})
Arrival	Minute	Instead of the whole day, the non-on-time query may focus on a certain time period within a day. The domain of "Arrival" is partitioned by minute. Thus, there are 1440 (24 × 60) subdomains in total for "Arrival"
Delay	Time interval	Non-serious delay can be allowed, and meaningful thresholds for the partitioning values need to be selected. The domain of "Delay" is partitioned into 10 one-minute partitions plus one extra partition for delays exceeding 10 min (e.g., {(0, 1), [1, 2), ..., [9, 10), $[10, \infty)$})

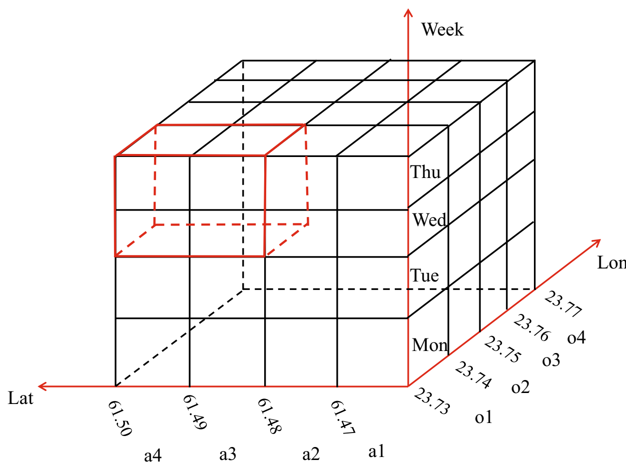


Fig. 4 An example of attribute combination query

Next, we introduce the details of parallel query computation by three typical kinds of queries. Please note that the queries to be discussed below are typical application scenarios for Bus-OLAP. However, it is easy to perform more complex queries using index operations with Spark. All bus non-on-time queries are performed using the framework shown in Fig. 5. Specifically, Bus-OLAP takes the query condition as input. In the query process, Bus-OLAP firstly maps the conditions into tuples. Secondly, for each condition tuple, Bus-OLAP conducts the bit operations on vectors using parallel computation with Spark. Finally, Bus-OLAP returns the results that satisfy the query conditions.

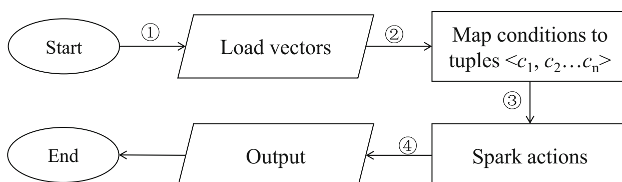


Fig. 5 Framework of distributed query and computing. There are four steps: ① loading vectors into memory according to the query conditions; ② mapping the conditions into tuples $<c_1, c_2, \ldots, c_n>$, where c_i, $0 \leq i \leq n$, is the query condition; ③ partitioning the data into nodes and performing the bit operations on vectors by Spark; ④ writing the output of the query result

3.4.1 Temporal Queries

The typical temporal queries include querying the number of non-on-time events over a period of time and querying the routes that have the most non-on-time events over a period of time.

Consider a query on the number of non-on-time events over a period of time, given time interval t, and non-on-time condition z (ahead of time, on-time, or delayed). The corresponding condition tuple in Fig. 5 is $<t, z>$. Please note that there may be many condition tuples due to the number of time intervals. We compute each condition tuple in a parallel, using the computational model of Spark. For each condition tuple, the query result is obtained by index operations on bit-vectors. The number of non-on-time events equals the number of bits set to 1 in the result vector.

Similarly, the query on the route that has the most non-on-time events over a period of time can be easily implemented. Given query conditions with respect to time interval t, non-on-time condition z, and bus line l, the computation on each condition tuple $<t, z, l>$ returns the number of delaying events. Then, the query result is the bus line that has the maximum number of delaying events over all tuple results.

3.4.2 Spatial Queries

There are some typical spatial queries, such as KNN and RKNN, that can be applied in the bus data in the following way. Given a bus stop q with a number of delaying events,

the analyst may want to know the bus stops close to q and analyze the factors related to the delays.

After partitioning attributes “Longitude” and “Latitude,” we get a division of space into a grid. Each bus stop is located in a grid. For a bus stop q, the main steps to find the k nearest bus stops are: (i) search the candidate bus stops by extending a q-centered rectangular space that consists of grids; (ii) once there are at least k bus stops contained in the rectangular space, calculate the distance from q to the kth nearest stop p (denoted by $\mathrm{dis}(q,p)$); (iii) find k nearest neighboring stops within a q-centered circle space with radius $\mathrm{dis}(p,q)$.

Example 7 Consider bus stop q in Fig. 6. To find five bus stops nearest to q, the first step is extending the q-centered rectangle. As R_1 is the first (smallest) rectangle containing 5 bus stops, and p_5 is the fifth nearest bus stop to q within R_1, stops located in the q-centered circle with radius $\mathrm{dis}(q,p_5)$ are the candidates for the query results (5 nearest neighboring stops to q).

Considering the characteristics of index operations, it is easy to find bus stops located within a rectangle area compared to a circle one. Thus, in Bus-OLAP, for stop q in Fig. 6, we find the KNN stops of q by checking the stops located within the smallest rectangle containing the q-centered circle with radius $\mathrm{dis}(q,p_5)$, i.e., R_2. Please note that it is easy to find stops located in R_2 by index operations based on R_1. For rectangle R_1 in Fig. 6, the index operation on Longitude $\mathrm{op}_{\mathrm{lon}} = \mathrm{OR}(\{o_2,o_3,o_4,o_5,o_6\})$ and on Latitude $\mathrm{op}_{\mathrm{lat}} = \mathrm{OR}(\{a_2,a_3,a_4,a_5,a_6\})$. After extending the rectangular area from R_1 to R_2, two partitions o_1 and o_7 are added into the Longitude of R_1. Similarly, the two partitions a_1 and a_7 are added into the Latitude of R_1. Consequently, the new result on Longitude is calculated as $\mathrm{op}_{\mathrm{lon}} \mid \mathrm{OR}(\{o_1,o_7\})$ and the new result on Latitude is calculated as $\mathrm{op}_{\mathrm{lat}} \mid \mathrm{OR}(\{a_1,a_7\})$, respectively. Obviously, the search area can be efficiently extended by performing index operations on bit-vectors iteratively.

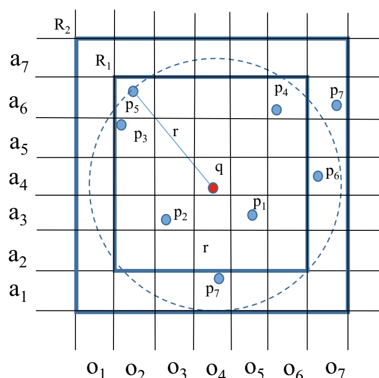


Fig. 6 An example of KNN query

The RKNN query can be used to find all bus stops whose KNN stops include the given bus stop q. We implement the query using parallel computation on Spark. In the framework of Fig. 5, the condition tuple is $< q,p,k >$, where q is the given bus stop. Spark is used to compute each tuple, returning the result indicating whether q is one of the KNN stops of p. The RKNN query result is a set of bus stops that satisfy the query condition.

3.4.3 Spatial–Temporal Queries

Detecting the spatial–temporal zone of bus delay aggregation can be done by querying the zone where delay aggregation occurs in a period of time. We measure the significance of spatial–temporal aggregation delay by log-likelihood ratio (LLR). Given a zone S and a time interval T, the log-likelihood of bus delay taking place, denoted as $\mathcal{L}(S,T)$, is

$$\mathcal{L}(S,T) = \begin{cases} \mathcal{D}(S,T)\times\log(r_1) + (\mathcal{D}(\widetilde{S},\widetilde{T}) - \mathcal{D}(S,T))\times\log(r_2) \\ -\mathcal{D}(\widetilde{S},\widetilde{T})\times\log\dfrac{\mathcal{D}(\widetilde{S},\widetilde{T})}{\mathcal{N}(\widetilde{S},\widetilde{T})}, \quad r_1 > r_2 \\ 0, \quad r_1 \le r_2 \end{cases} \tag{2}$$

where

$$r_1 = \frac{\mathcal{D}(S,T)}{\mathcal{N}(S,T)}, r_2 = \frac{\mathcal{D}(\widetilde{S},\widetilde{T}) - \mathcal{D}(S,T)}{\mathcal{N}(\widetilde{S},\widetilde{T}) - \mathcal{N}(S,T)}$$

where $\widetilde{S}$ and $\widetilde{T}$ are the maximal zone and maximal time interval, respectively; S and T are the observed zone and time interval, respectively; $\mathcal{N}(S,T)$ denotes the total number of bus arrivals within S and T; and $\mathcal{D}(S,T)$ denotes the number of delaying events within S and T.

Our target is to find the zone during a time interval that has maximal LLR in bus delay aggregation analysis. Figure 7 illustrates the processing of the delay aggregation query with Spark. Step 1 in Fig. 7 corresponds to Step 2 in Fig. 5, and Steps 2 and 3 in Fig. 7 correspond to Step 3 in Fig. 5. Bus-OLAP takes query conditions as input. Firstly, Step 1 joins the query conditions into condition tuples $< o,a,d,t,de >$, where the variables denote “Longitude,” “Latitude,” “Date,” “Time,” and “Delay,” respectively. Secondly, Step 2 partitions tuples to nodes of the Spark cluster and gets the new tuples of LLR. Finally, Step 3 searches the tuple with maximal LLR, which is the query result.

3.5 Query with Data from Different Sources

The on-time rate of bus service is vulnerable to the impact of external factors. For example, the delaying events happen more frequently in rainy days and foggy days. Clearly,

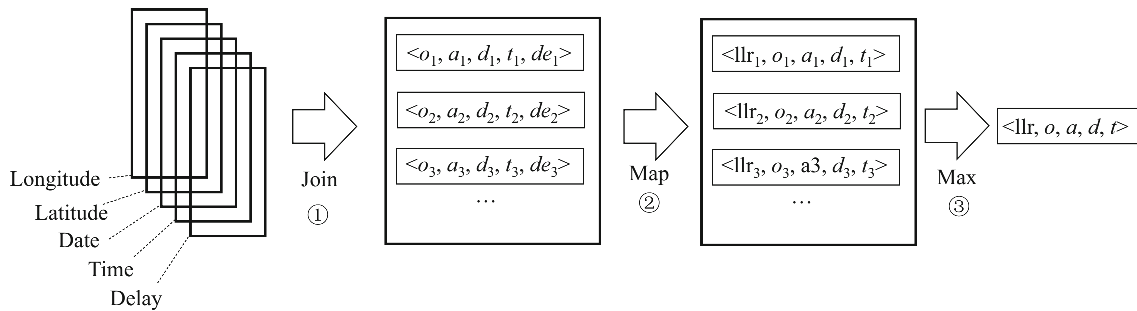


Fig. 7 Spark process of delay aggregation query

it is interesting and useful to consider various external factors, e.g., weather, when performing bus non-on-time analysis. To this end, we introduce a flexible implementation in Bus-OLAP such that queries containing conditions on external factors can be efficiently performed.

It is challenging to query with external factors, since the external factors are extracted from different sources of data. For example, the bus journey records shown in Table 1 do not contain any weather information. If users want to investigate the impact of weather on bus delay, weather information should be collected at first. Technically, there are two steps to support queries containing data from different sources.

– The first step is fusing external factors with bus journey records. In other words, more attributes with respect to external factors are added to bus journey records.
– The second step is building indexes for the external factors in order to support efficient query.

Considering that changes in weather may significantly impact the bus on-time rate, we collect weather data from *Weather Underground*, which is a Web site sharing weather information of global cities.[2] Next, we introduce our method to fuse external factors with bus journey records for bus non-on-time queries by taking weather data as an example.

To give an example, we collect Tampere's local *average daily temperature* and *daily weather activity* from *Weather Underground*. The reasons we collect these two factors are that (i) *average daily temperature* and *daily weather activity* are two main features of weather status, and (ii) the data types of these two factors are typical. The domain of *average daily temperature* is continuous, while the domain of *daily weather activity* containing "rain," "fog," and "thunder" is enumerable. (Here, we only consider the weather activities that affect the bus on-time rate negatively.)

Please recall that, as listed in Table 1, the bus journey records contain temporal information (i.e., date and bus arrival time). As both *average daily temperature* and *daily weather activity* are time related, it is easy to associate each bus journey record with the average temperature and weather activity in that moment.

Example 8 On August 5, 2015, at Tampere city, the average daily temperature was 15 °C and the weather was rainy. Then, for the 3rd record listed in Table 1, we fuse it with weather factors (in bold font) as follows.

Stop	Name	Lat.	Long.	Line	Date	Arrival	Delay	**Temp**	**Activity**
3084	Kuoppa maentie	61.4799	23.8046	560	2016-08-05	00:09:48	−34	**15**	**Rain**

For the sake of query efficiency, it is necessary to build indexes for the external factors. Similar to building indexes for bus journey data (Sect. 3.2), for each external factor, we build corresponding indexes based on the domain partition. For an external factor whose data type is continuous (e.g., *average daily temperature*), the index is built like building the index on "Delay". Specifically, for each bus journey record, we construct a bit-vector indicating the interval that the value of *average daily temperature* locates in. For an external factor whose data type is enumerable (e.g., *daily weather activity*), the index is built like building the index on "day of the week". Specifically, we use a set of bit-vectors to indicate the values of *daily weather* for all bus journey records.

[2] https://www.wunderground.com.

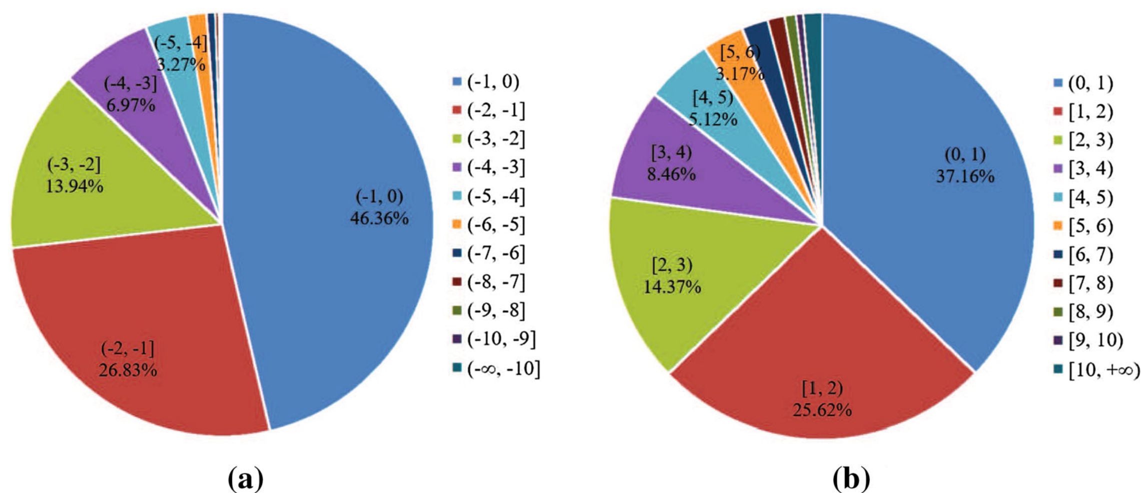


Fig. 8 Distributions of the delay (minutes) of non-on-time arrivals, **a** early case, **b** delaying case

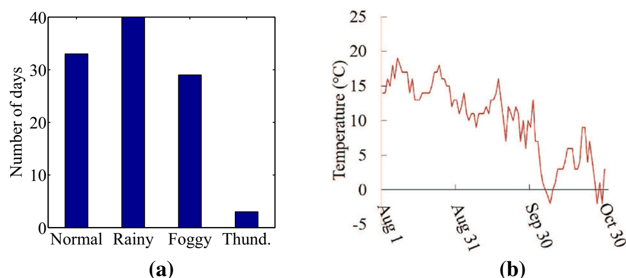


Fig. 9 Statistics on daily weather activities and average daily temperature change **a** weather activities and **b** temperature change

4 Empirical Evaluation

In this section, we report a systematic empirical study on a real-world bus data set from Tampere city.[3] The data set includes the bus stop information of Tampere, 43 bus routes, 75 bus stops of a line on average, and 6,297,520 records generated from all weekdays from August 1, 2015, to October 30, 2015. We also collected daily weather activity and average daily temperature for each record from *Weather Underground* Web site.

All algorithms are implemented in Java and compiled using JDK 7. The distributed environment is built using Spark 1.6.2 and includes eight nodes. Each node is a computer with an Intel Core i7-6700 3.40 GHz CPU, and 64 GB main memory, running Ubuntu 14.04 operating system.

Figure 8 illustrates the quantitative relationships between the numbers of non-on-time arrivals with different time intervals (measured in minutes) ahead or delayed. From Fig. 8, it is interesting to see that most (over 75%) of non-on-time arrivals happened within ± 3 min compared to the scheduled time. Specifically, for the early case, there are over 87% arrivals arriving slightly ahead of schedule. And for the delaying case, there are 77% arrivals arriving slightly behind the schedule. Considering the complex situations in the real world, it is impractical to expect that every bus arrives on-time exactly. Naturally, we treat bus arrivals with slight time ahead or delayed as normal (on-time) cases. Thus, in our empirical study, we label each bus arrival as a non-on-time event including “early case” or “delaying case”, if the value of “Delay” is < -3 min (“early case”) or > 3 min (“delaying case”).

Figure 9a illustrates the number of days with respect to certain weather. The daily weather contained in weather data include: rain, fog, and thunder. We can see that the rainy weather is more frequent than other weather activities. As the number of days with thunder (i.e., 3) is too small, we only consider rain and fog in this empirical study. Figure 9b shows the change of daily temperature. The highest temperature is 19 °C on August 15, and the lowest temperature is -2 °C on October 9, 28, and 30. There are 51 such days that the average daily temperature is above 10 °C.

4.1 Effectiveness

We verify the effectiveness of Bus-OLAP using three kinds of typical queries: temporal queries, spatial queries, and spatial–temporal queries. For the temporal queries, there are two frequent queries: (1) What is the number of delaying events in every weekday? (2) Which are the routes that have the most delaying events?

Figure 10 illustrates the numbers of non-on-time arrival events on weekdays in August, September, and October, respectively. As we stated in Sect. 1, the main concern of bus non-on-time query is the delaying case, and in Fig. 10,

[3] http://trafficdata.sis.uta.fi.

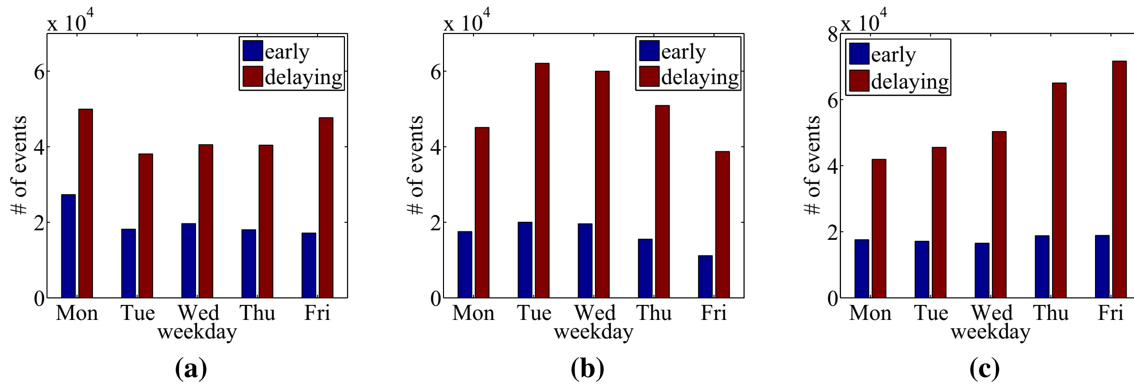


Fig. 10 Number of non-on-time events **a** Aug, **b** Sept, and **c** Oct

Table 3 Bus lines with the largest number of delaying events during different time intervals

Month	Time interval	Bus line	# Of arrival events	# Of delaying events	Rank
August	[6:00, 10:00)	13	32,626	6373	1
		28	21,539	4335	2
		29	25,682	2788	3
	[10:00, 14:00)	13	31,952	6050	1
		28	19,882	3691	2
		3	37,188	3532	3
	[14:00, 18:00)	13	37,530	12,041	1
		3	44,279	7989	2
		8	36,115	7775	3
	[18:00, 22:00)	13	30,953	5367	1
		3	29,992	4598	2
		28	19,522	2684	3
September	[6:00, 10:00)	13	34,921	6158	1
		28	19,124	5457	2
		29	30,467	4732	3
	[10:00, 14:00)	13	31,660	5662	1
		8	29,489	4043	2
		11	14,940	3636	3
	[14:00, 18:00)	13	37,817	12,186	1
		8	39,397	9151	2
		3	43,333	7885	3
	[18:00, 22:00)	13	30,328	6577	1
		3	30,831	5458	2
		17	20,973	2658	3
October	[6:00, 10:00)	3	39,093	4886	1
		28	15,134	4214	2
		13	31,022	4166	3
	[10:00, 14:00)	8	33,193	5907	1
		3	43,794	4341	2
		13	28,877	4080	3
	[14:00, 18:00)	8	44,679	11,490	1
		13	35,443	10,165	2
		3	48,568	9529	3
	[18:00, 22:00)	3	33,152	7019	1
		13	25,796	4831	2
		8	18,234	3976	3

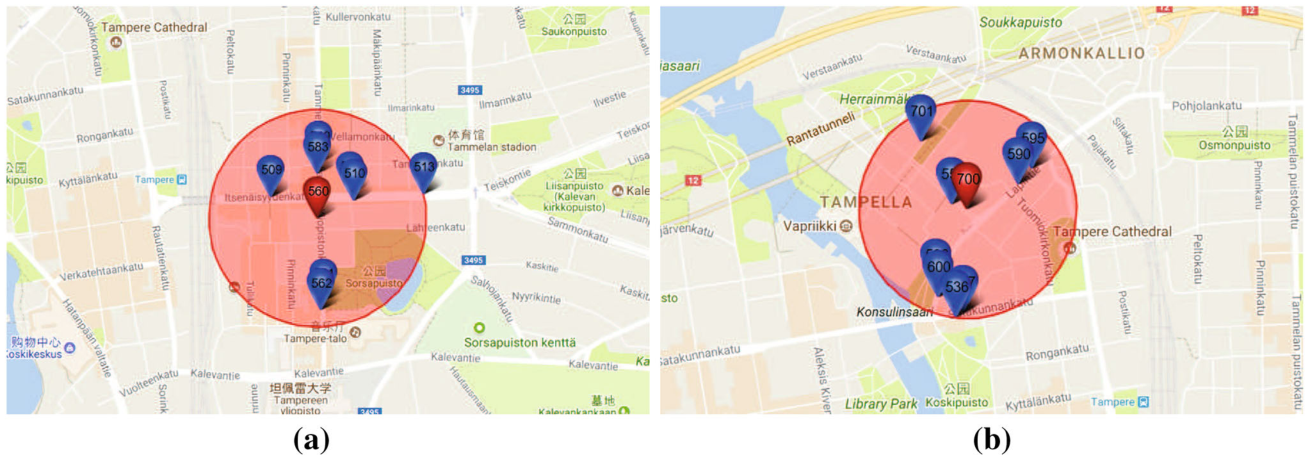


Fig. 11 Illustration of KNN queries ($k = 8$), **a** stop "Yliopistonkatu", **b** stop "Villa Viola"

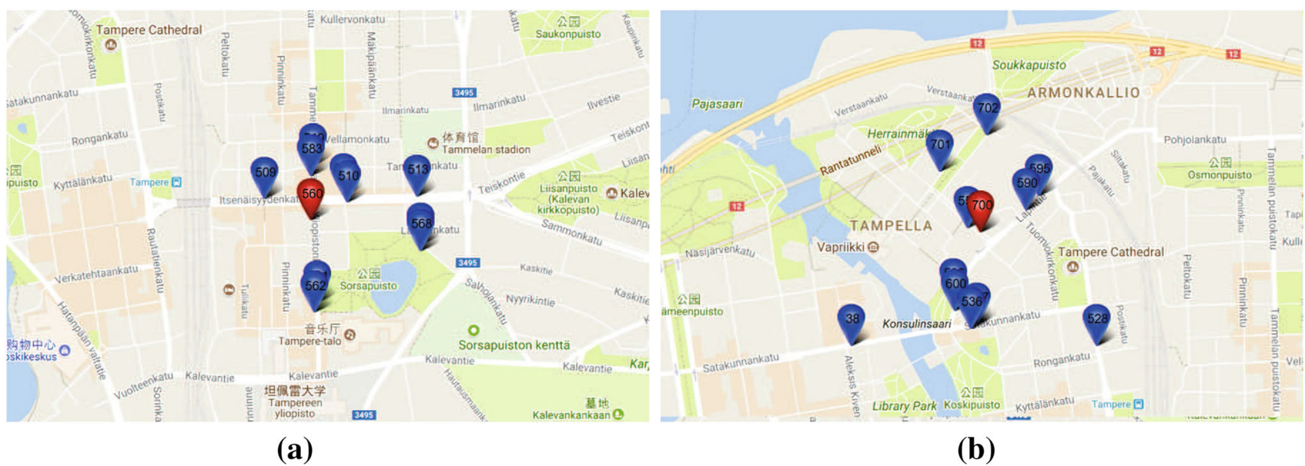


Fig. 12 Illustration of RKNN queries ($k = 8$), **a** stop "Yliopistonkatu", **b** Stop "Villa Viola"

we can see that the number of delaying cases is much larger than the number of early cases. It is interesting to see that the weekday with the maximum number of non-on-time events keeps changing from August to October. For example, in August, the number of delaying cases on Monday is the largest, in September, the largest number is on Tuesday, while in October, the largest one is on Thursday.

Table 3 lists the top 3 routes where delaying events occur frequently during different time intervals. It is interesting to see that Lines 13, 8, and 3 are the routes with the largest numbers of delaying events, especially in time interval [14:00, 18:00), over the three months. We also find that the lines that have frequently delaying events during different periods are different. One possible explanation is that the direction of crowd flow in the morning rush hour is different from that in the evening rush hour. The buses were delayed due to the getting in and getting off time by large numbers of passengers.

Figures 11 and 12 illustrate the query results of spatial queries, i.e., KNN and RKNN, on two bus stops, respectively. The two target bus stops are "Yliopistonkatu"(ID: 560) and "Villa Viola" (ID: 700). In Fig. 11, the red points are the query bus stops, while the blue points are the eight bus stops nearest to the query stops. Similarly, in Fig. 12, the red points are the query bus stops, while the blue points are the RKNN bus stops for the query stops.

One typical bus non-on-time query is detecting bus delay aggregation. Please recall that, as listed in Table 2, we partitioned the space of Tampere city into 2000×1000 equal-sized grids in Bus-OLAP. Let the maximum search range be 100×100 grids. Then, by Bus-OLAP, we can answer following questions with different given time intervals: (1) "On which weekday the most significant delay aggregations took place?", (2) "In which time interval the most significant delay aggregation occur?", (3) "Which day and time interval has the most significant bus delay aggregations."

Table 4 Zone with spatial–temporal delay aggregation of every weekday

Weekday	Time interval	Zone (minLon, maxLon, minLat, maxLat)	LLR
Mon	[6:00, 10:00)	(23.810540, 23.844397, 61.503022, 61.517329)	185.55
	[10:00, 14:00)	(23.675500, 23.713204, 61.531799, 61.547732)	94.24
	[14:00, 18:00)	(23.592783, 23.630487, 61.502534, 61.518467)	1264.80
	[18:00, 22:00)	(23.597015, 23.632410, 61.515378, 61.530336)	285.78
Tue	[6:00, 10:00)	(23.752831, 23.791304, 61.474408, 61.490666)	279.73
	[10:00, 14:00)	(23.709356, 23.713973, 61.503672, 61.505623)	89.35
	[14:00, 18:00)	(23.592399, 23.630872, 61.502209, 61.518467)	896.02
	[18:00, 22:00)	(23.597015, 23.631641, 61.515541, 61.530173)	285.07
Wed	[6:00, 10:00)	(23.752831, 23.791304, 61.474571, 61.490829)	325.94
	[10:00, 14:00)	(23.675500, 23.707048, 61.531799, 61.545130)	86.39
	[14:00, 18:00)	(23.761680, 23.800153, 61.476847, 61.493105)	1338.30
	[18:00, 22:00)	(23.597015, 23.632410, 61.515378, 61.530336)	300.43
Thu	[6:00, 10:00)	(23.752062, 23.788996, 61.472620, 61.488227)	226.48
	[10:00, 14:00)	(23.675500, 23.707048, 61.531799, 61.545130)	127.99
	[14:00, 18:00)	(23.761680, 23.800153, 61.476847, 61.493105)	1024.59
	[18:00, 22:00)	(23.597015, 23.632410, 61.515378, 61.530336)	331.23
Fri	[6:00, 10:00)	(23.812079, 23.844397, 61.503022, 61.516679)	125.07
	[10:00, 14:00)	(23.595861, 23.628948, 61.504485, 61.518467)	167.20
	[14:00, 18:00)	(23.761680, 23.800153, 61.476847, 61.493105)	1914.59
	[18:00, 22:00)	(23.604710, 23.643183, 61.496356, 61.512614)	397.30

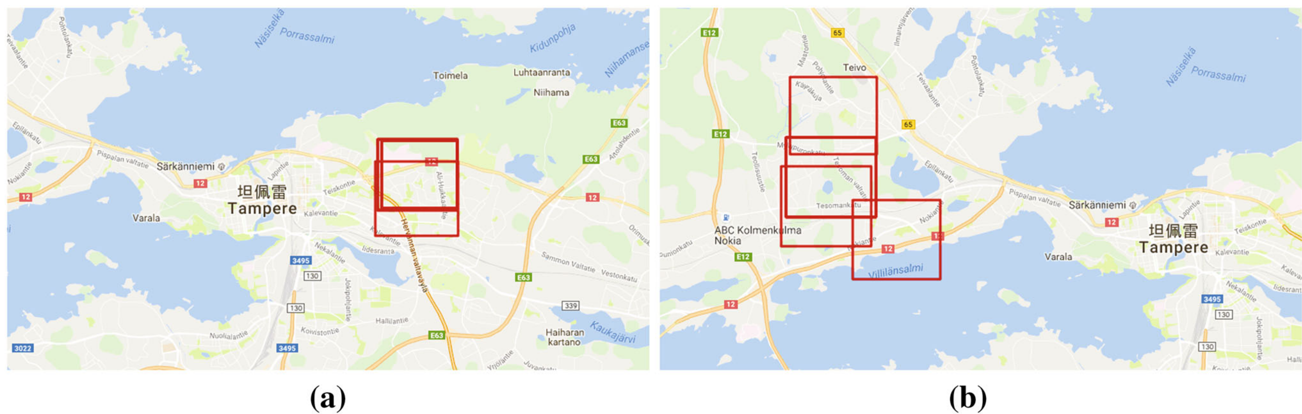


Fig. 13 Illustration of zones with significant bus delay aggregation, **a** morning, **b** afternoon

Table 4 lists the zones of spatial–temporal delay aggregation with different time intervals. We present a zone in the form of (*minLon*, *maxLon*, *minLat*, *maxLat*), where *minLon* and *maxLon* denote, respectively, the minimal and maximal longitude of a zone, *minLat* and *maxLat* denote, respectively, the minimal and maximal latitude of a zone. As listed in Table 4, there are more significant delay aggregations during [14:00, 18:00) compared to other time intervals. Furthermore, period [14:00, 18:00) on Friday is the worst time interval as it has the largest number of significant delay aggregations. We guess it is because the traffic flow is heavy in the afternoon rush hour of Friday.

Figure 13 shows the zones with significant bus delay aggregation in the morning (a) and in the afternoon (b), respectively. We find that the zones where bus delay aggregations happened are non-overlapping. We think one possible reason is that the zones where buses are full of people are different in the morning and afternoon. To verify the correctness of the detected zones with bus delay aggregation, we consider the heat delaying cases in Fig. 14. It is easy to see that the detected zones are consistent with the heat distributions.

Intuitively, the bus delay aggregation is closely related to the traffic flow, detecting the zones with significant bus delay

Fig. 14 Bus delay heat map, **a** morning, **b** afternoon

Fig. 15 Zones with the most significant busy delay aggregation w.r.t. weather activity in morning and evening rush hours. (The detected zones in normal weather, rainy weather, and foggy weather are, respectively, indicated by rectangles with red solid line, blue dot line, and dark dash line.), **a** [07:00, 08:00), **b** [08:00, 09:00), **c** [16:00, 17:00), **d** [17:00, 18:00)

aggregation can reflect the change of traffic flow. In addition, it is interesting to investigate the impacts of weather conditions on the bus delay aggregation. To this end, we perform queries taking weather factor into consideration over morning and evening rush hours, respectively.

In Fig. 15, we use rectangles with red solid line, blue dot line, and dark dash line to indicate the zones where significant bus delay aggregation happened in normal weather, rainy weather, and foggy weather, respectively. We can observe the flow of traffic changes from the east side of

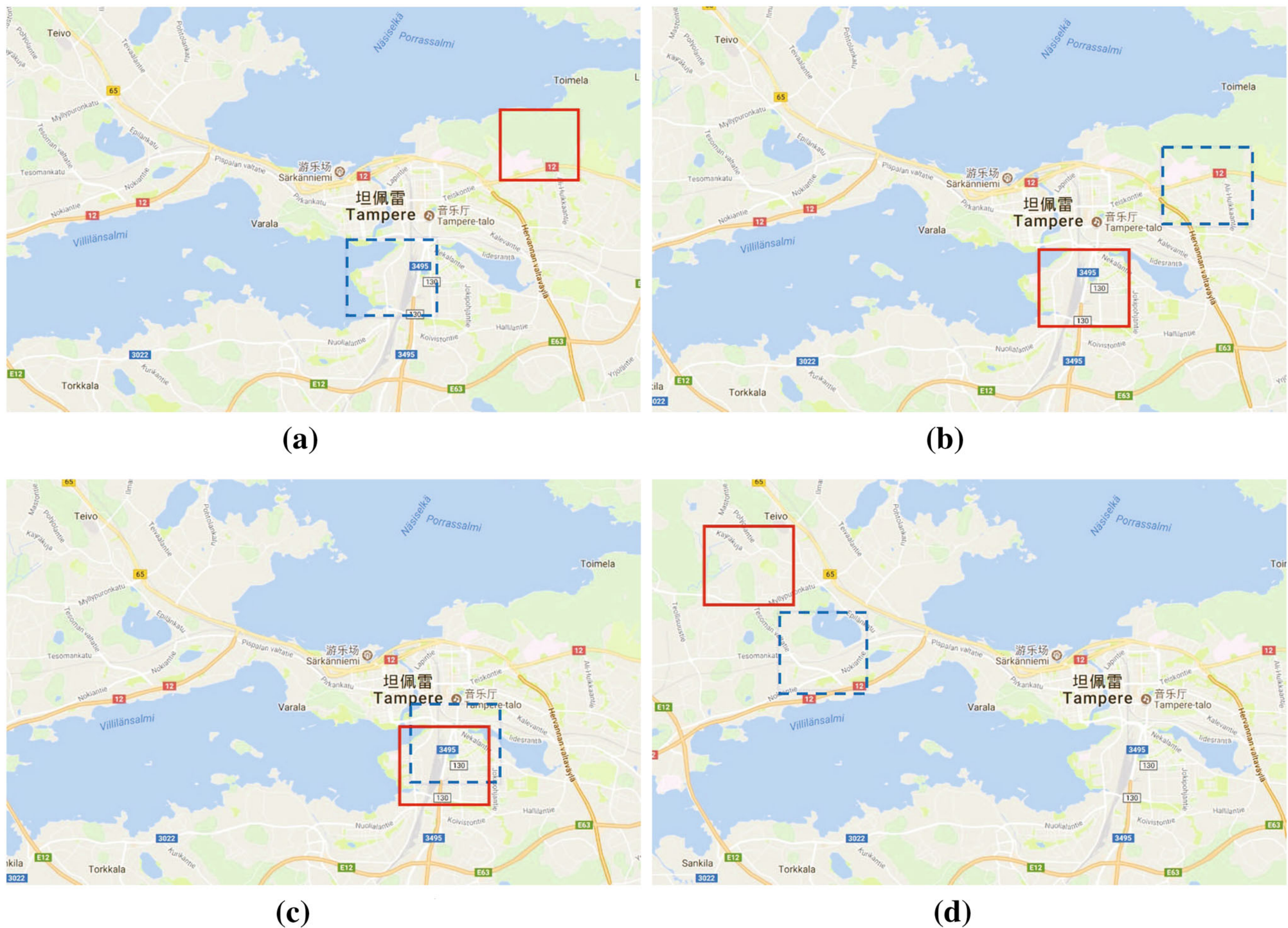


Fig. 16 Zones with the most significant busy delay aggregation w.r.t. average daily temperature in morning and evening rush hours. (Rectangles with red solid line are the zones when the average daily temperature $> 10\,^{\circ}$C, and rectangles with blue dash line are the zones when the average daily temperature $\leq 10\,^{\circ}$C.), **a** [07:00, 08:00), **b** [08:00, 09:00), **c** [16:00, 17:00), **d** [17:00, 18:00)

Tampere city to the west side in both normal weather and foggy weather from 16:00 to 18:00. We also see that the zone with the most significant bus delay aggregation in foggy weather is clearly different from the ones in normal and rainy weather during 8:00 $\sim$ 9:00 am. In addition, we can see that the zone with the most significant bus delay aggregation in rainy weather is clearly different from the ones in normal and foggy weather during 17:00 $\sim$ 18:00. We guess the reason lies in that, firstly, the impact of fog on bus delay aggregation mainly exists in the morning, since fog disperses in the afternoon, and secondly, the impact of rain mainly exists in the evening due to the accumulation of rainwater on the road surface.

In Fig. 16, we illustrate the zones with the most significant busy delay aggregations when the average daily temperature is above $10\,^{\circ}$C (rectangle with red solid line) and is not above $10\,^{\circ}$C (rectangle with blue dash line), respectively. Clearly, the zones detected under different temperature ranges are different. As shown in Figs. 15 and 16, taking weather factors into bus non-on-time analysis is necessary and reasonable.

4.2 Efficiency

In Bus-OLAP, we build the indexes based on bit-vectors and perform bus non-on-time queries by index operations. In order to verify the effectiveness of the index building, we first test the runtime for building the indexes with respect to the number of records in Bus-OLAP. Then, we compare the query efficiency using Bus-OLAP against the query using relational database MySQL 5.5. The results of efficiency test are shown in Fig. 17. We test the efficiency by three most frequently used queries, querying the time interval with the most delaying events for temporal query shown in Fig. 17b, searching the RKNN bus stops for spatial query illustrated in Fig. 17c and finding the zones with significant bus delay aggregation from Monday to Friday for temporal–spatial query shown in Fig. 17d. Clearly, as shown in Fig. 17a, we can see that the index

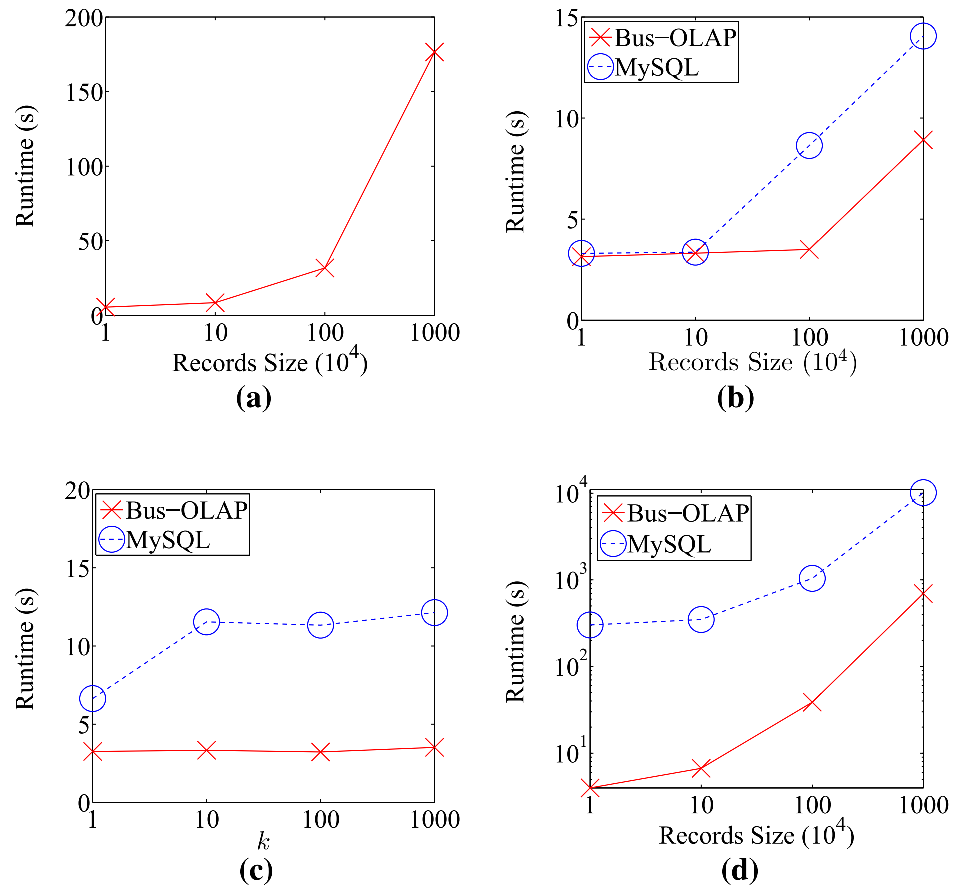


Fig. 17 Efficiency on index building and query, **a** index building, **b** query on time interval with most delay, **c** RKNN query ($q = 560$), **d** query on zones with delay aggregation

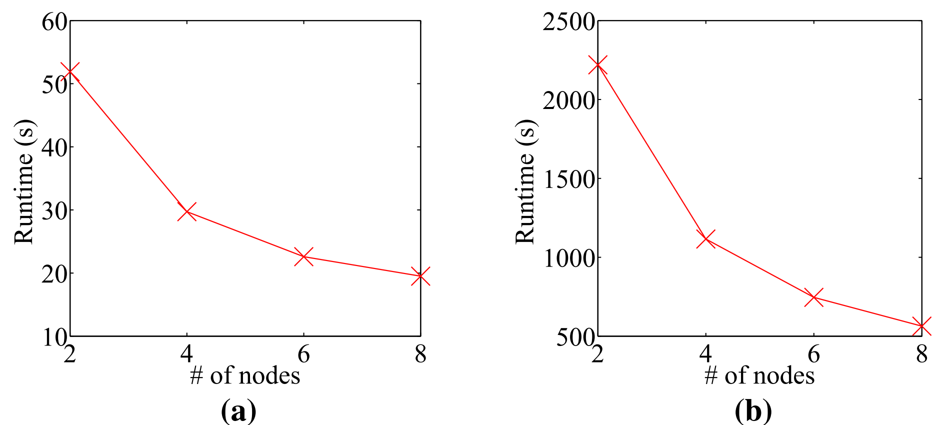


Fig. 18 Query efficiency w.r.t. number of computing nodes, **a** computing the LLR of each grid, **b** query on delay aggregation during [16:00, 17:00)

building in Bus-OLAP is efficient, and the building time increases linearly with the size of bus journey records. Moreover, performing bus non-on-time query by Bus-OLAP is significantly more efficient than by MySQL.

To satisfy the requirement of large number of queries, we have implement Bus-OLAP on Spark, so that parallel query computation can be supported. Again, we test the query efficiency with respect to the number of computing nodes by finding the zones with significant bus delay aggregation during a certain time interval in weekday. Figure 18a shows the runtime for computing the *LLR* of every grid with respect to the number of computing nodes. Figure 18b shows the runtime for querying delay aggregation zones during time interval [16:00, 17:00) on weekdays. It is clear that in both LLR computation and aggregation queries, the runtime of Bus-OLAP decreases when more computing nodes are used.

In summary, by our proposed index building method and the parallel query framework, Bus-OLAP is efficient for bus non-on-time query.

5 Conclusions

In this paper, we tackled the novel and interesting problem of non-on-time query over bus journey data. We designed a model, named Bus-OLAP, to support non-on-time queries over the data. For the sake of efficiency, we built the index of bus journey data using bit-vectors and introduced index operations to convert the queries into bitwise operations. In addition, we implemented the distributed query computation based on the Spark framework. Our experiments verified the effectiveness and efficiency of Bus-OLAP.

There are several interesting issues that deserve research effort in the future. First, we will consider more complex scenario applications of non-on-time analysis and fuse some implicit factors (e.g., road conditions, number of passengers) that affect the bus. Second, there are large bus data sets in reality, and we try to apply the Bus-OLAP to more real data and more data sources of other cities. Then, we will continue on optimizing the frequently used non-on-time queries. It is also interesting to consider dynamic computation of the current traffic situation, that is, analyzing if the current situation is different from “normal.” Moreover, we plan to combine the bus data analysis with the management of urban traffic and study the relationships between them.

Acknowledgements The authors are grateful to Dr. Paula Syrjärinne for her help on bus journey data preparation and to the editor and the anonymous reviewers for their constructive comments, which have helped to improve this paper.

Open Access This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

References

1. Bao J, He T, Ruan S, Li Y, Zheng Y (2017) Planning bike lanes based on sharing-bikes’ trajectories. In: Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining, pp 1377–1386
2. Chen M, Liu Y, Yu X (2015) Predicting next locations with object clustering and trajectory clustering. In: Proceedings of the 19th Pacific–Asia conference on knowledge discovery and data mining, part II, pp 344–356
3. Eldawy A, Mokbel MF (2013) A demonstration of spatialhadoop: an efficient mapreduce framework for spatial data. Proc VLDB Endow 6(12):1230–1233
4. Ghosh B, Basu B, O’Mahony M (2009) Multivariate short-term traffic flow forecasting using time-series analysis. IEEE Trans Intell Transp Syst 10(2):246–254
5. Guttman A (1984) R-trees: a dynamic index structure for spatial searching. In: Proceedings of ACM SIGMOD’84, pp 47–57
6. Han Y, Moutarde F (2016) Analysis of large-scale traffic dynamics in an urban transportation network using non-negative tensor factorization. Int J Intell Transp Syst Res 14(1):36–49
7. Jagadish HV, Ooi BC, Tan KL, Yu C, Zhang R (2005) iDistance: an adaptive B^+-tree based indexing method for nearest neighbor search. ACM Trans Database Syst 30(2):364–397
8. Kong X, Xu Z, Shen G, Wang J, Yang Q, Zhang B (2016) Urban traffic congestion estimation and prediction based on floating car trajectory data. Future Gener Comput Syst 61:97–107
9. Liu D, Chen H, Qi H, Yang B (2013) Advances in spatiotemporal data mining. J Comput Res Dev 50(2):225–239
10. Pang LX, Chawla S, Liu W, Zheng Y (2011) On mining anomalous patterns in road traffic streams. In: Proceedings of the 7th international conference on advanced data mining and applications, part II, pp 237–251
11. Pang T, Duan L, Nummenmaa J, Zuo J, Zhang P (2017) Bus-OLAP: a bus journey data management model for non-on-time events query. In: Proceedings of the 1st international joint conference on web and big data, part II, pp 185–200
12. Sistla AP, Wolfson O, Chamberlain S, Dao S (1997) Modeling and querying moving objects. In: Proceedings of the 13th international conference on data engineering, pp 422–432
13. Stathopoulos A, Karlaftis MG (2003) A multivariate state space approach for urban traffic flow modeling and prediction. Transp Res Part C: Emerg Technol 11(2):121–135
14. Syrjärinne P, Nummenmaa J (2015) Improving usability of open public transportation data. In: Proceedings of the 22nd ITS world congress, pp 5–9
15. Syrjärinne P, Nummenmaa J, Thanisch P, Kerminen R, Hakulinen E (2015) Analysing traffic fluency from bus data. IET Intell Transp Syst 9(6):566–572
16. Ting RH, De Almeida T, Ding Z (2006) Modeling and querying moving objects in networks. VLDB J 15(2):165–190
17. Wang Y, Papageorgiou M, Messmer A (2007) Real-time freeway traffic state estimation based on extended Kalman filter: a case study. Transp Sci 41(2):167–181
18. Wu X, Duan L, Pang T, Nummenmaa J (2016) Detection of statistically significant bus delay aggregation by spatial–temporal scanning. In: Proceedings of APWeb 2016 workshops, pp 277–288
19. Xia D, Li H, Wang B, Li Y (2016) A map reduce-based nearest neighbor approach for big-data-driven traffic flow prediction. IEEE Access 4:2920–2934
20. Xie X, Xiong Z, Hu X, Zhou G, Ni J (2014) On massive spatial data retrieval based on spark. In: Proceedings of WAIM 2014 international workshops, pp 200–208
21. Yu X, Pu KQ, Koudas N (2005) Monitoring k-nearest neighbor queries over moving objects. In: Proceedings of the 21st international conference on data engineering, pp 631–642
22. Yuan J, Zheng Y, Zhang L, Xie X, Sun G (2011) Where to find my next passenger. In: Proceedings of the 13th international conference on ubiquitous computing, pp 109–118
23. Zaharia M, Xin RS, Wendell P, Das T, Armbrust M, Dave A, Meng X, Rosen J, Venkataraman S, Franklin MJ, Ghodsi A, Gonzalez J, Shenker S, Stoica I (2016) Apache spark: a unified engine for big data processing. Commun ACM 59(11):56–65
24. Zhang J, Zheng Y, Qi D (2017) Deep spatio-temporal residual networks for citywide crowd flows prediction. In: Proceedings of the 31st AAAI conference on artificial intelligence, pp 1655–1661
25. Zheng Y, Capra L, Wolfson O, Yang H (2014) Urban computing: concepts, methodologies, and applications. ACM Trans Intell Syst Technol 5(3):38