Public Transport (2020) 12:631–645
https://doi.org/10.1007/s12469-020-00248-8

ORIGINAL RESEARCH

Efficient monitoring of public transport journeys

Felix Gündling[1] · **Florian Hopp**[1] · **Karsten Weihe**[1]

Accepted: 18 August 2020 / Published online: 12 September 2020

Abstract
Many things can go wrong on a journey. From minor disturbances like a track change to major problems like train cancellations, everything can happen. The broad availability of smartphones enables us to keep the traveler up-to-date with information relevant for the journey. This way, the traveler can react to changes as early as possible and make well-informed decisions. Naive approaches are too inefficient to monitor a large number of journeys in real-time. This paper presents an efficient way to monitor millions of journeys in parallel. In our approach, the selection of change notices to be communicated to a traveler may be flexibly adapted to the travelers individual needs.

Keywords Real-time · Public transport · Personalized · Connection monitoring

1 Introduction

Every day, millions of travelers use public transportation to get to their destinations. Not all of those journeys run smoothly. Problems may be caused by delays, reroutings, cancellations, track changes, etc. If they occur, information is key to finding a solution. The earlier a problem is communicated by the transportation provider, the more options are available to the customer to react. With the advent of smartphones, it is now possible to inform the user as soon as new information about the situation becomes apparent. This imposes some constraints on the data processing: once a real-time update (e.g. a delay) is available, the system needs to determine the

This work was partially supported by Deutsche Bahn AG.

✉ Felix Gündling
guendling@cs.tu-darmstadt.de

Florian Hopp
hopp@cs.tu-darmstadt.de

Karsten Weihe
weihe@cs.tu-darmstadt.de

[1] Technical University of Darmstadt, Hochschulstraße 10, 64289 Darmstadt, Germany

affected journeys in a timely manner. Due to the large number of travelers, the real-time monitoring of all current and future (i.e. booked) journeys is a challenging task for the transportation provider.

On the one hand, the customers always want to be up-to-date regarding the status of their journey. On the other hand, no one wants to annoy customers with journey updates they are not interested in: while some customers might be interested in a notice about an upcoming transfer, others only want to be bothered with essential information regarding the feasibility of the journey. Here, the remaining time until the corresponding event (e.g. the transfer in question) is important: a traveler is probably not interested in a predicted change in arrival time of 2 min when there are still 2 h left until the interchange.

This would probably trigger many unnecessary alerts, since forecasts so far in the future are inherently inaccurate. Not only currently active journeys need to be monitored. Changes may also concern all future journeys already booked by customers. For example, a schedule change due to planned construction work may change the arrival time at the destination, which should be communicated to all affected customers who already booked their journey.

In this paper, we address both of the above-mentioned challenges: our system is capable of monitoring millions of customer journeys in real-time on commodity hardware; every customer can set an individual profile for each journey. This profile specifies precisely about which changes they want to get informed. Separate profiles for each journey, not one per customer, enables customers to specify a different "alert level" for different journeys. For example, for their way to work they are only interested in cancellations and delays above 15 min but when traveling to an important appointment, they might choose a more elaborate setting.

This paper is organized as follows. Section 2 discusses related approaches. Section 3 introduces the setting and notation. Section 4 describes the configuration options the monitoring profile provides. Section 5 explains the input real-time data and how it is processed. Section 6 presents two monitoring approaches which are being evaluated with real-world data (timetables as well as real-time updates from Deutsche Bahn AG) in Section 7.

2 Comparison to related work

Previous work (Frede et al. 2008; Müller-Hannemann and Schnee 2009) focuses on incorporating real-time information into a graph model which represents the public transport schedule. This way, it is possible to provide feasible real-time alternatives to customers. However, Frede et al. (2008); Müller-Hannemann and Schnee (2009) do not address any topics related to personalized connection monitoring. In contrast, our approach builds on top of an up-to-date real-time graph and provides a scalable but personalized connection monitoring.

As shown in recent studies conducted in Chicago (Tang and Thakuriah 2012) and New York (Brakewood et al. 2015), even information in the form of a real-time arrivals boards may increase ridership and customer satisfaction. A study from

Stockholm by Cats et al. (2011) suggests that real-time information may change the paths chosen by passengers.

The system presented in this paper takes real-time information one step further: instead of providing information statically through displays at stations or on customer request in a smartphone application, the system monitors all customer journeys. Based on a personalized monitoring profile, it informs customers (e.g. through smart phone push notification, SMS, or email) about the events they are interested in.

The monitoring system presented in this paper can work hand in hand with disturbance management systems (e.g. the ones described in Jespersen-Groth et al. 2009; Törnquist 2007): efficiently detecting all travelers who are affected by a decision (e.g. a track change) and informing them is a core functionality of our approach. A seamless and timely communication in case of disturbances improves the outcomes for the passenger.

3 Basic terminology

We introduce the notion of departure events e_{dep} and arrival events e_{arr}. Important events are events where the traveller either enters or exits a vehicle. Monitoring these events is sufficient to monitor all relevant aspects (e.g. feasibility) of a journey. We define an interchange as a pair of an arrival and a departure event $(e_{\text{dep}}, e_{\text{arr}})$ where both events have to take place either at the same station or at two stations in convenient walking distance. As our system allows walks between nearby stations to occur in connections, the station(s) of these events do not need to match.

For each station, the schedule provides a minimal transfer time $\text{tt}(s)$ required to get from one vehicle to another. Depending on the input data, this function can be fine-grained and adjustable for eldery or handicapped people. However, for our evaluation we did not have access to data at this level of detail. Furthermore, the schedule contains stations between which it is possible to walk: this is defined as a tuple (s_1, s_2, t) where s_1 and s_2 are two stations and t is the corresponding time to walk from s_1 to s_2.

Thus, an interchange $(e_{\text{dep}}, e_{\text{arr}})$ between two vehicles at the same station s is considered valid if $t(e_{\text{dep}}) - t(e_{\text{arr}}) \geq \text{tt}(s)$. For foot walks, the estimated walking time between the two stations may not be undercut.

4 Monitoring profile

All interchanges as well as the first departure and the last arrival of a connection are particularly important for the user. Besides the changes concerning the time of an event which is part of an interchange, users may be concerned about the time between the arrival and the departure of an interchange. More specifically, they might not want to be informed about changes where departure and arrival of an interchange are both delayed by N minutes: they have still the same time buffer for the interchange. However, they could be interested in a delayed arrival

when the connecting train departs on schedule because this would make the interchange inconvenient or even infeasible. Note that both aspects can be configured separately. Psychologically, it might feel safer to get a message when any of those aspects change—which is also supported by the system described here.

The system stores the attribute value (e.g. the timestamp of an interchange event, the interchange time buffer, the overall feasibility of the journey, etc.) most recently communicated to the users for each attribute for each aspect of the stored connection. The initial values are taken from the journey stored by each user. Usually, these are the scheduled values. However, in case of a real-time alternative for a missed connection, the initial values may differ from the scheduled values. If the latest value announced to the users differs from the actual real-time value more than a specified threshold, the users get informed about this change. After an update has been sent to them, the updated value will be stored by the system. The system allows monitoring the following attribute value changes (relative to the value most recently announced to the user) separately:

1. For the first departure: later departure, earlier departure
2. For the last arrival: later arrival, earlier arrival
3. For the events of an interchange: earlier arrival, later arrival, earlier departure, later departure, and the time difference between departure and arrival
4. For an interchange: a notice about the oncoming interchange a fixed time before the arrival
5. Messages to customers using a specific vehicle: these messages address specific stops of the stop sequence of a vehicle.

The monitoring profile is comprised of one two-dimensional look-up table for each aspect described above.

Another important factor for the decision of whether to inform users about a change is the remaining time until the event will take place. The closer to the event in question, the more urgent it is to inform the users and the more precise predictions can be. For example, an anticipated delay increase of 2 min might be relevant 30 min before the actual event. However, predicted delays in a complex public transport network (e.g. the German railway network) cannot be very precise on an extended time horizon of several hours and are thus subject to fluctuations. We introduce the following rule set as means to avoid informing users about changes that are not (yet) relevant and subject to change: the system requires a two-dimensional look-up table where for each remaining time until the actual event (row) and change in minutes (column), it is specified whether the users should get informed about that change. Note that this is not necessarily the way, an end-user would need to specify this: a mobile app might come completely pre-configured or condense these detailed settings into a few profiles to suite different user groups.

Changes in different directions (increase of delay/decrease of delay) might be of different importance to the user. Consequently, the system incorporates different matrices for each value (listed above) for each direction. For example, a

change of five minutes delay increase might have a larger impact for the arrival event of an interchange than for the departure event.

5 Real-time data

5.1 Identifier

Real-time data (for example provided by the operator—in our case Deutsche Bahn) contains data that references *stations*, identified by their unique station id. Later on, we will make use of the following entities:

- *vehicles*, identified by the train number,[1] first stop id, and first stop departure time
- *events*, identified by the vehicle (see above), the station the event takes place, its schedule time and the event type (either arrival or departure)

This data will be used as key in our data structures (e.g. a hash map) to uniquely identify the referenced entity.

5.2 Message types

In this section, we will outline the structure of the real-time data processed by the system:

- A delay message contains either anticipated or real (measured) delays of public transport vehicles. It refers to the vehicle in question as well as a list of events with their corresponding delay (which can be zero or even negative if the event took place earlier than planned).
- A rerouting message refers to a specific vehicle and contains a list of added stops with their corresponding schedule time and a list of removed stops.
- A cancellation message is analogous to rerouting messages, except that there are no added stops.
- A track change message refers to a specific event and contains the new track, the vehicle arrives/departs at.
- A free text message contains a text written by the operator and targets a specific section (identified by the first and the last stop) of a vehicle. It can be used to inform the travelers about important issues (e.g. guidance in critical situations).

[1] The train number by itself is only unique for one traffic day.

5.3 Delay propagation

The information basis for the connection monitoring is the schedule timetable on the one hand and a continuous stream of real-time messages on the other hand. The timetable is represented as a time-dependent routing graph (as introduced in Disser et al. (2008)). This graph is updated according to the received real-time data to represent the current and predicted real-time situation. The basic update approach is described in Müller-Hannemann and Schnee (2009). This approach propagates delays along the vehicle path, respects waiting time rules among vehicles and fixes resulting data inconsistencies (e.g. delay update messages that permute the event order of a vehicle). Note that the application of waiting time rules enables the propagation of delays from one vehicle to another. This way, delays can spread not only along the vehicle path but throughout the entire network.

One objective of the system is to warn users about anticipated problems, not those that are already obvious from the real (measured) delays from the past. The real-time data stream does not contain the propagated delays. Thus, a monitoring approach based on the real-time data itself will not work. Therefore, the system uses both, delays as well as propagated delays for the connection monitoring.

6 Connection monitoring

In this section, we will discuss the two approaches to monitor a set of stored connections. Section 7 evaluates and compares the performance of both approaches.

6.1 Periodic approach

One method to monitor a set of stored connections (regarding the values listed in Sect. 4) is to check each connection separately in a fixed period of time (e.g. every minute). The first step is to check for each connection whether it is affected by any received real-time change. The second step is to check if this real-time change needs to be communicated to the users based on their individual monitoring profile (introduced in Sect. 4). Later on, we will refer to this approach as the *periodic* approach. But since most stored connections will not have relevant changes that need to be communicated to the user, this may result in a waste of runtime.

6.2 Event-based approach

The basic idea of our event-based approach is to determine the set of connections that need to be checked based on the events that changed during the real-time update of the routing graph. Assuming that not every real-time update affects every event contained in the stored connections, this is much less computational effort compared to the periodic approach. This assumption is reasonable considering that most stored (i.e. booked) connections will not take place right now or in the next few hours. Even

if the periodic approach would only check the connections of the current, previous, and next day (to detect problems in overnight connections), the system would need to check many connections that are not actually affected by the real-time update.

6.2.1 Handling cancellations, reroutings, delays, and track changes

To efficiently determine the set of connections affected by incoming real-time updates, we utilize data structures which allow for a fast lookup. Our goal is to create a function δ_{ev} that takes a set of updated events (either updated directly or through propagation) R as input and returns the subset of these connections that is affected by at least one of the changes. For every changed event e, we aim to implement a lookup of affected stored connections $E[e]$ where E is an efficient $O(1)$ lookup from an event to a set of connections (i.e. a hash map data structure). Using the information that uniquely identifies an event as described in Sect. 5.1 as key, we can build a data structure E mapping an event to the set of connections that contain this event. Regarding the algorithmic complexity, a hash map is a sound choice for this task. With the help of this lookup table E, we can implement $\delta_{\text{ev}}(R) = \bigcup_{e \in R} E[e]$.

To monitor the properties described in Sect. 4 (besides the free text messages—the handling of which will be described in Sect. 6.2.2), it is sufficient to check the interchanges of a journey as well as its first and last stop. Instead of monitoring every event contained in every stored connection, we can focus on $2N_i + 2$ events per journey where N_i is the number of interchanges $(e_{\text{dep}}, e_{\text{arr}})$ of journey i. This accounts for all interchanges as well as the first departure and last arrival.

Since a single connection i is associated with several keys ($2N_i + 2$ events) which wastes memory, we create an indirection: the map just stores a unique integer that references the associated connection. To lookup the details of a connection, we introduce an additional map data structure mapping from the unique connection integer to the details (stop sequence, used vehicles, walk times, etc.) describing the associated connection. When adding a new connection to the set of monitored connections, this connection needs to be indexed: every important event (first departure, last arrival, and all interchanges) gets extracted and added to the mapping.

6.2.2 Handling free text messages

For the special case of free text messages, the approach described in Sect. 6.2.1 is not suitable because free text messages can target arbitrary events in the journey, not just interchanges, the first departure, and last arrival. Fortunately, instead of indexing all events of the journey, indexing the vehicles used in a journey is sufficient. Analogously to E which maps events to connections, hash map V maps vehicles onto a set of connections containing the respective vehicle. Thus, we can define a lookup function for vehicles $\delta_{\text{v}}(T) = \bigcup_{t \in T} V[\mathit{vehicle}(t)]$ where T is a set of free text messages and $\mathit{vehicle}(t)$ extracts the vehicle in question from a free text message. Uniting the results of both functions, δ_{ev} and δ_{v} yields the set of affected connections.

6.2.3 Putting the pieces together

After the affected connections have been determined (through δ_{ev} and δ_{v}), the next step is to check for each connection whether the change exceeds the configured threshold introduced in Sect. 4. Thus, the system checks every aspect mentioned in the monitoring profile (described in Sect. 4). Basically, the system needs to store the value known to the user. For example, as described in Sect. 4, for each interchange, the system stores the arrival time, the departure time and the interchange buffer. For each stop, it stores the free text messages that were communicated to the user. This way, it is possible to check whether the user needs to be informed: if a connection is affected by a real-time change (which is determined using δ_{ev} and δ_{v}), each stored value is compared to the current value. Since the time until the event will take place is also known, both row (change in minutes) and column (remaining time until the actual event) of the change within the two-dimensional look-up table (introduced in Sect. 4) are available. Consequently, the system can compare the current change in each entity with the threshold value from the two-dimensional look-up table.

6.2.4 Deferred checking

If the difference between the value (e.g. interchange time buffer for a specific interchange of the connection) last known to the user and the current value determined by the system (based on the real-time schedule and delay propagation) exceeds the threshold from the two-dimensional look-up table, the system informs the user instantly. Otherwise, the system needs to check the connection again later: as time goes by, the time until the monitored event decreases. Therefore, other (smaller) threshold values become relevant in the two-dimensional look-up table. By iterating the rows (time until event) in the column (computed change) from the current row in decreasing order, we can determine the first row where the user would need to get informed if the computed change stays the same. Therefore, the system may setup a timer to check the connection again at this time. This can be seen as an event-driven simulation. Since these are potentially many timers, the system collects all theses events in a queue which is sorted by the timer expiry (earliest check first). Since all users have their own personal preferences on time thresholds, each event may occur multiple times (one for each stored connection). This way, only one timer for the first element in the queue is required. When the timer expires, the system iterates all entries from the queue until the first entry where the expiry value is not exceeded anymore. When checking those iterated entries it may be the case that the user already was informed because of another change that occurred between the insertion into the vector and the iteration. It may also be the case that the user needs to be informed. In both cases, the entry is removed from the vector. Lastly, the value may have changed between the insertion into the vector and the iteration so that the user does not need to get informed now. In the last case, the entry will be reinserted into the queue with the new timer expiry value. Comparing the event-based approach to the periodic approach, the event-based approach reduces the number of connections to look at by checking on demand, not periodically.

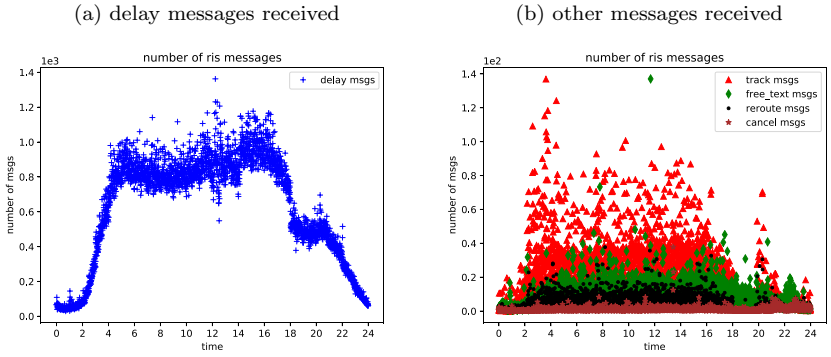

Fig. 1 Distribution of real-time (delays, cancellations including reroutings, track changes and free text updates) messages over the day: one dot represents all messages that occurred in a 30 s time period. **a** Shows delay messages received from Deutsche Bahn. **b** All other message types without the delay messages

7 Evaluation

In this section, we present an experimental study of the concepts presented in Sect. 6. We evaluate the event-based version and the periodic version. Additionally, we measure the runtimes of parallel versions of both implementations. All versions are implemented in C++ and run on a machine with an Intel® Core™ i7-6850K CPU and 64GB of RAM.

7.1 Schedule timetable and real-time data

Both, the schedule timetable as well as the real-time data are provided by Deutsche Bahn. For our evaluation, we use the full public transport schedule timetable of Germany. This includes buses, streetcars, subway, suburban trains, regional as well as long-distance trains. We analyze a timespan of one week. In this timespan, 250,000 stations are served with an average of 24 M events per day. The time-dependent timetable graph has 4.9 M nodes and 15.1 M edges. A cumulative real-time update is sent every 30 s by Deutsche Bahn. Note that our system (especially the event-based approach) is also capable of handling updates at arbitrary times or update periods. Every real-time update package sent by Deutsche Bahn contains an average of approximately 666 real-time updates (1.9 M updates per day). Those updates cover all trains (including suburban railway) operated by Deutsche Bahn as well as the real-time data of certain local public transportation authorities (e.g. for Berlin or the Rhein/Ruhr area). The system propagates these primary delays through the timetable network which results in another 6713 forecasted delays per update that need to be processed by the connection monitoring system.

Figure 1 shows the delay distribution over a regular Tuesday (number of messages against time of day). Note that a delay message is not only sent for delayed arrival and departure events but also for events that take place as scheduled. Thus, there is at least one delay update for each event of vehicles operated by Deutsche

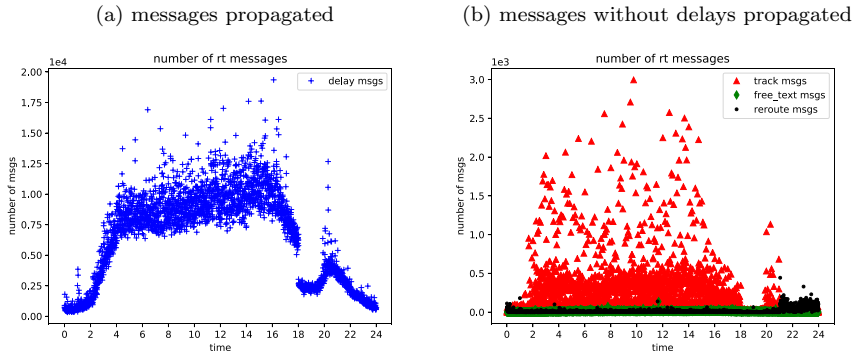

Fig. 2 Distribution of real-time (delays, cancellations including reroutings, track changes and free text updates) messages over the day: one dot represents all messages that occurred in a 30 s time period. **a** Shows all messages including those propagated by our system. **b** Shows all messages without the delay messages including those propagated by our system

Bahn plus several delay forecast updates that need to be processed by the system. Each dot represents the messages from a 30 second time interval. As is clearly visible, the delays are not uniformly distributed but raw delays received from Deutsche Bahn (left plot) dip to below 100 update messages (per 30 s) at night times while staying above 900 update messages every 30 s for several hours in the daytime. The reason for this is that Deutsche Bahn operates more trains at daytime than at night time. Peaks at 1200 messages to process in one turn are possible and should not lead to any disruptions. Message spikes are caused by systems that are not under our control. Thus, the reasons for these deviations are not transparent to us. The plot in the middle shows the number of total delays including raw delays as well as propagated delays. Here, we can see that propagation adds roughly one order of magnitude to the message count: raw messages (left plot) peak around 1200 messages every 30 s against 12,500–17,500 when propagating delays. The number of other messages (track change messages, free text messages, and reroute messages) is comparatively small. The pattern of decreased message volume at night times is visible here, too (Fig. 2).

7.2 Performance Comparison

In this section, we will analyze both approaches presented in Sect. 6 (periodic and event-based) regarding the number of required check operations as well as runtime performance. For this, we generated one million connections by using a multi-criteria shortest path routing algorithm. Source, destination, and departure time interval are chosen randomly with a uniform distribution over all stations in the schedule and the complete schedule period of 1 week. These connections were stored and indexed by both approaches: for the periodic approach no further processing is required, whereas for the event-based approach all important events need to be indexed. Event indexing took 19 min for 1 M journeys. However, note

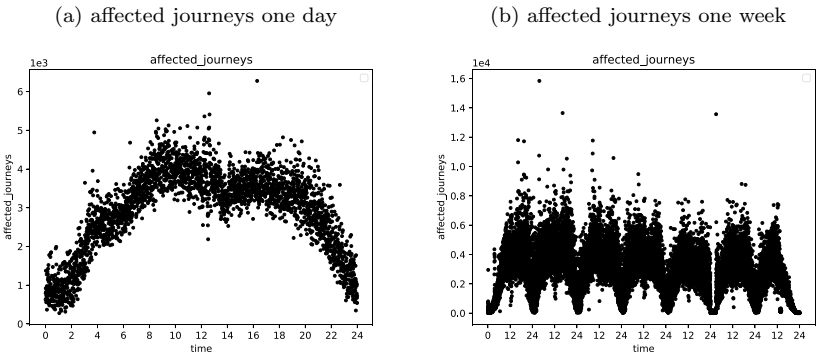

Fig. 3 The number of connections affected by real-time changes over time for the event-based approach

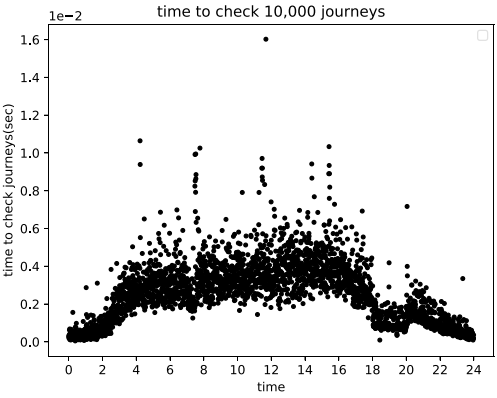

Fig. 4 Runtimes of the event-based connection checking approach in 30 s batches (as received by Deutsche Bahn)

that this needs to be done only once and usually happens gradually when users book their journeys. Indexing a single journey takes 1.15 ms on average.

Using the periodic approach, all one million connections need to be checked in every iteration. Figure 3 shows the number of connections affected by real-time messages (either directly received from Deutsche Bahn or generated by delay propagation). Note that if a connection is affected by a delay, this does not necessarily mean that the user needs to get informed. As we can see, from the one million stored connections, only approximately 10,000 connections need to be checked more closely.

Figure 4 shows the runtime of the event-based approach over the day. The increased message volume at daytime compared to nighttime results in increased runtimes at daytime. The runtime of the periodic approach is roughly constant at close to 10 s per check run. The efficient lookup data structures used in the

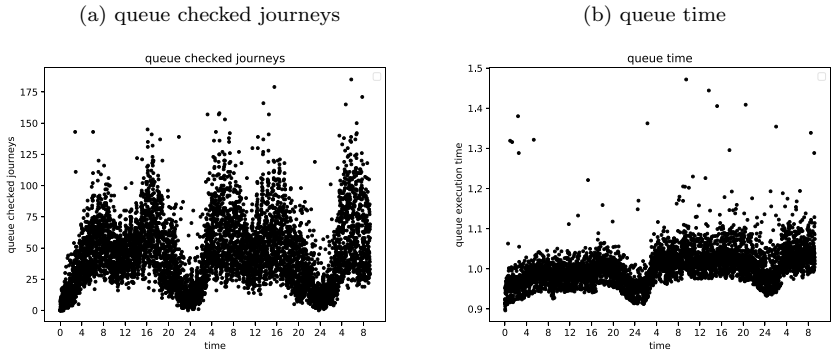

Fig. 5 Timer Queue for 1 Million stored Connections over two days. **a** Shows the queue size, **b** shows the journeys, which are checked again and **c** shows the time the queue needed for execution

event-based approach lead to a runtime reduction of approximately two orders of magnitude at daytimes (0.1 s against nearly 10 s).

7.3 Timer queue performance

As described in Sect. 6, the system keeps a list of connections that need to be checked again at some point in the future. Since the timetable data provided by Deutsche Bahn has a granularity of one minute, the system only works at discrete points in time. Figure 5 shows (a) the number of checked journeys and (b) the runtime at each minute of the day for two consecutive days. As there are less trains operated at night times, the pattern of reduced work load at night times is similar to those visible in Fig. 4.

7.4 Improvements

The performance of both approaches can be further improved through better resource usage of the underlying hardware. An analysis of better memory usage (RAM in addition to disk) and CPU usage (multi-core instead of single-core) follows.

On-Disk against In-Memory Basically, all connections need to be stored persistently so they can be loaded after system restarts. This on-disk database can be used for the connection monitoring, too. Faster access times can speed up the checks. The difference is shown in Fig. 6. The evaluation was conducted with different numbers of connections (100, 1 k, 10 k, 100 k, 1 M). Obviously, both approaches perform better with in-memory storage. While the periodic approach is less influenced by the storage medium, the event-based approach is very sensible in this regard: for 10,000 connections, the performance does not show a big difference. However, for 100,000 connections and 1,000,000 connections the runtimes leap. This difference in behavior may be explained by the access patterns of both approaches: the periodic

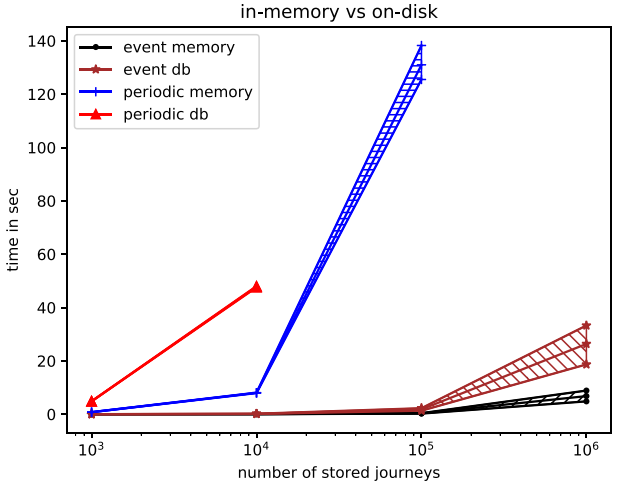

Fig. 6 Analysis of how the storage medium (on-disk against in-memory) influences the runtime performance (25% quantile, median, and 75% quantile) of both approaches for different counts of stored connections

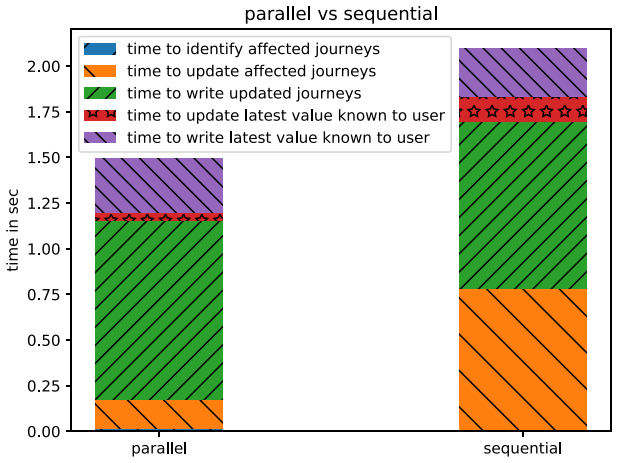

Fig. 7 Different parts and their time requirements

approach on the one hand iterates all connections sequentially. On the other hand, the event-based approach does not access connections in a particular order. Thus, the event-based approach benefits more from the fast random-access read performance of RAM storage. Finally, we can see that in practice, the periodic approach scales poorly and is not able to keep up with the real-time message input (one update every 30 s vs. more than 2 min for each check) for 100,000 registered connections. The event-based approach scales easily up to 1 M connections.

Parallelization Modern server CPUs have many cores. Utilizing all cores is essential to make use of the compute resources the hardware provides. Both approaches presented in this paper are parallelizable in a straightforward manner: the checks for each real-time update (going through the map lookup data structures/iterating all connections) do not depend on each other. Therefore, several stages of the connection monitoring can be processed in parallel. Figure 7 shows the results of an evaluation conducted on a 6-core CPU: overall the average time required to process one real-time update was reduced more than 25%. Since the persistent storage medium does not benefit from a multi-core CPU, all I/O bound processes (time to write updated journeys as well as the time to write the latest value known to the user) do barely benefit from the parallelization. However, all computational steps (identifying and updating affected journeys as well as updating the latest value known to the user) can be performed much faster.

8 Conclusion and outlook

We presented two different approaches to monitoring booked connections of public transport travelers: the naive approach checks every stored connection periodically whereas the event-based approach reduces the number of connections that need to be checked by several magnitudes (e.g. from 1M to below 10,000). The system provides an elaborate rule system to configure which information should be pushed (e.g. via an mobile application, SMS, or e-mail) to the user (on a per-connection basis) to avoid annoying the users with updates they are not interested in. Our evaluation, which is based on real data provided by Deutsche Bahn (schedule timetable as well as real-time updates), shows that the event-based system is capable of monitoring millions of connections on a default desktop workstation. Different improvements (parallel real-time update processing as well as in-memory storage) further enhances the runtime performance of the system. In our future work, we aim to incorporate real-time updates of other means of transportation (e.g. flight data or traffic flow updates) into our system.

Funding Open Access funding provided by Projekt Deal.

References

Brakewood C, Macfarlane GS, Watkins K (2015) The impact of real-time information on bus ridership in New York City. Transp Res Part C Emerg Technol 53:59–75

Cats O, Koutsopoulos H, Burghout W, Toledo T (2011) Effect of real-time transit information on dynamic path choice of passengers. Transp Res Record J Transp Res Board 2217:46–54

Caulfield B, O'Mahony M (2009) A stated preference analysis of real-time public transit stop information. J Public Transp 12(3):1–20

Disser Y, Müller-Hannemann M, Schnee M (2008) Multi-criteria shortest paths in time-dependent train networks. In: McGeoch CC (eds) Experimental algorithms. WEA 2008. Lecture Notes in Computer Science, vol 5038. Springer, Berlin, pp 347–362

Frede L, Müller-Hannemann M, Schnee M (2008) Efficient on-trip timetable information in the presence of delays. In *OASIcs-OpenAccess Series in Informatics*, vol 9. Schloss Dagstuhl-Leibniz-Zentrum für Informatik. https://doi.org/10.4230/OASIcs.ATMOS.2008.1584

Jespersen-Groth J, Potthoff D, Clausen J, Huisman D, Kroon L, Maróti G, Nyhave Nielsen M (2009) Robust and online large-scale optimization. Disruption management in passenger railway transportation. Springer, New York, pp 399–421

Müller-Hannemann M, Schnee M (2009) Efficient timetable information in the presence of delays. In: Ahuja RK, Möhring RH, Zaroliagis CD (eds) Robust and online large-scale optimization. Lecture Notes in Computer Science, vol 5868. Springer, Berlin, pp 249–272

Tang L, Thakuriah PV (2012) Ridership effects of real-time bus information system: a case study in the city of Chicago. Transp Res Part C Emerg Technol 22:146–161

Törnquist J (2007) Railway traffic disturbance management—an experimental analysis of disturbance complexity, management objectives and limitations in planning horizon. Transp Res Part A Policy Pract 41(3):249–266