MOBANA: a Distributed Stream-based Information System for Public Transit

Tianyi MA, Gianmario MOTTA, Kaixu LIU, Dongmeng LIU
Dept. of Industrial and Information Engineering
University of Pavia
Pavia, Italy
motta05@unipv.it, {tianyi.ma01, kaixu.liu01, dongmeng.liu01}@ateneopv.it

Abstract—**Public transit generates a wide range of diverse data, which include static data and high-velocity data streams from sensors. Integrating and processing this big real-time data is a challenge in developing analytical systems for public transit. We here propose MOBANA (MOBility ANAlyzer), a distributed stream-based system, which provides real-time information to a wide range of users for monitoring and analyzing the performance of public transit. To do so, MOBANA integrates the diverse data sources of public transit, and converts them into standard and exchangeable data formats. In order to manage such diverse data, we propose a layered architecture, where each layer handles a specific kind of data. MOBANA is designed to be efficient. E.g., it identifies the real time position of vehicles by adjusting planned position with real-time data as needed, thus dropping network load. MOBANA is implemented by Distributed Stream Processing Engine (DSPE) and Distributed Messaging System (DMS), which pursue scalable, efficient and reliable real-time processing and analytics. MOBANA was deployed as pilot in Pavia, and tested with real data.**

Keywords-public transit; destributed stream processing; distributed messaging system; GTFS; real-time analytics

I. INTRODUCTION

Public transit is critical in cities. Thanks to the availability of a variety of public transit data (e.g., transit schedule, routes, and sensor data), cities can develop analytic systems to monitor the performance of public transit, by modelling, integrating and analyzing real data rather than by simulating [1]. The analytical results on these data can serve municipalities, transit agencies, third-party organizations (e.g., researchers) and citizens [2]. However, designing such analytic systems requires integrating heterogeneous data and processing high-velocity data streams. Let us shortly illustrate these points.

A. Integration of heterogeneous data

Public transit data include several sources, namely static data published by transit agencies (e.g., timetables, routes, service alerts) and real-time data from sensors (e.g., vehicle positions from on-board GPS, real-time delay feeds). These data are typically independent and follow diverse formats (e.g., XML / JSON, CSV, PDF). Such diversity of data (i.e., the Variety and Veracity in big data) makes integration a critical task. Integration actually requires a great effort to understand the semantics of each source, and to solve semantic ambiguity, instance representation ambiguity, and data inconsistency [3]. Furthermore, because of the heterogeneity of source schemas, source data shall be unified and integrated into a same schema and be stored into a warehouse. Of course, traditional data integration and data warehouse techniques are insufficient to meet such needs. [4] From our viewpoint, a universal and efficient approach to integrate data into a universal standard is needed. General Transit Feed Specification (GTFS) [5] is the most widely used format for mapping static transit data. Until November, 2015, 965 transit agencies from more than 350 cities worldwide published their static data in GTFS format [6]. 25 transit agencies published open transit feeds in GTFS-Realtime, i.e., the real-time extension of GTFS [7]. However, until March, 2013, only 27% of public transit agencies in United States published their own open data [8]. That low percentage is explained by the GTFS conversion workload. Actually, mapping data into GTFS is time consuming, especially for the agencies in large cities who deal with large and complex transit networks. Furthermore, transit agencies have to re-map their GTFS data when transit schedules change.

B. High-velocity data stream

Public transit involves high-velocity data streams from sensors, e.g., GPS. The data stream can be used by numerous transportation services for traffic flow analysis, trip planning, geographical social networking, smart driving, and map matching [9]. However, because of the real-time nature of sensor data, the amount of data may grow exponentially [10]. For example, in Dublin, transit sensors generate 4-6 gigabytes real-time data per day [11]. Thus, an effective, and scalable processing model for real-time data is needed.

C. The MOBANA system

To overcome the above issues, we have developed an information system for real-time transit performance monitoring and analytics with heterogeneous data, named MOBANA (MOBility ANAlyzer) to cover the whole lifecycle of transit data, namely collection, conversion, processing, analysis, visualization, and sharing. In order to cover such cycle, a complete tool chain has been developed. MOBANA is based on a scalable distributed stream-based publisher and subscriber architecture, which combines Distributed Stream Processing Engine (DSPE) and Distributed Messaging System (DMS) to provide an efficient and reliable real-time processing. A set of Key Performance Indicators (KPIs) are defined, and accordingly, a KPI dashboard is implemented to monitor KPIs according to various dimensions (e.g., date, daytime, stops, routes, transit mode). The key contribution of MOBANA concerns a new hybrid approach, which combines both static data (in GTFS) and real-time data stream (in GTFS-Realtime) to analyze and visualize KPIs and vehicle positions in real-time. This approach reduces network traffic and server load versus the traditional approaches, which use raw GPS record. Finally, MOBANA publishes public transit information and analytic results to stakeholders by web services.

In this paper, section 2 illustrates the MOBANA framework and approach to the data lifecycle. Section 3 presents the case study of Pavia Mobility Analyzer. Section 4 discusses the performance of MOBANA. Section 5 compares similar systems, frameworks and approaches. Finally, section 6, “Conclusion”, summarizes achievements and sketches future improvements.

II. MOBANA

Public transit involves numerous stakeholder classes, which, consume transit data, as municipality, transit agencies, and citizens. In the typical usage scenario of MOBANA, municipalities and transit agencies analyze real-time mobility status through Key Performance Indicators (KPI) on a dashboard (Figure 6) or on a public transit map (Figure 7). These KPIs include the prediction of vehicle delay, the average vehicle delay by stop. In turn, citizens search, on the web or smart phone, open information about public transit (e.g., timetables, real-time bus delays, expected arrival time). Finally, third party organizations can use open information as a service to offer value added services or further analytics.

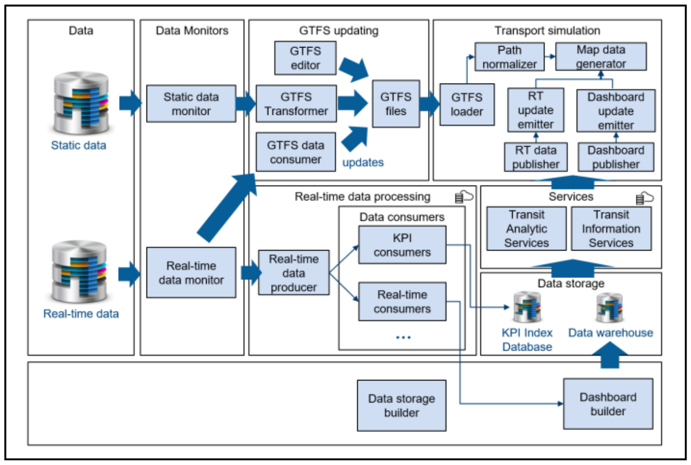


Figure 1. Components of MOBANA

MOBANA covers by specific components for all phases of the data lifecycle, namely gathering, conversion, processing and storage, analysis, visualization and sharing (Figure 1). In the following sections we describe the details of these phases and corresponding components.

A. *Data Gathering and Conversion*

MOBANA integrates static and real-time data sources. Let us shortly illustrate them.

Static transit data (time tables, routes, trips, agency, calendar, stops) are published by transit agencies in CSV or PDF format. These data can be also gathered from databases of public agencies. We implemented a GTFS Converter to convert these static data to GTFS format, which defines timetables, trips, routes, stops and associated geographic information for multiple public transit modes, e.g., bus, tramway and metro. Furthermore, the Static Data Monitor (SDM) listens to the changes in transit database. If changes are detected, SDM will trigger the GTFS Converter to reconstruct GTFS files. Finally, the GTFS Loader, the unique interface to access static data, will load these files for further processing and analysis.

In order to support transit agencies which do not have their GTFS files nor store their timetables and shape data (i.e., waypoints in routes), a GTFS Editor was implemented. It is a WYSIWYG (What You See Is What You Get) editor for creating, editing, and validating static transit data. Especially, it enables transit agencies to visually add and edit shape information on the map (see Figure 2). To the best of our knowledge, it is the first application that supports interactive editing and visualization of GTFS files.

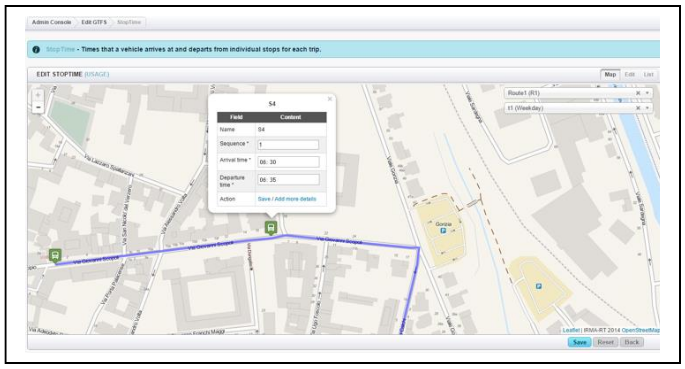


Figure 2. GTFS Editor

Real-time data (JSON / XML format) are gathered from data dispatching APIs of public transit agencies. MOBANA also supports GTFS-Realtime compliant transit feeds that contains at least one of following properties:

a) Trip time updates, which contains prediction of delays (in second) and Estimated Time of Arrival (ETA) and Estimated Time of Departure (ETD) of corresponding trip and stop;

b) Vehicle position updates, the real-time geo-location of given vehicle. A data producer receives these real-time data, converts them to GTFS-Realtime feeds and sends the feeds to message queues. The Trip Update Consumers and Vehicle Update Consumers get these data for further processing. In the next section we describe details of data producer and consumers;

c) Service alerts which indicate incidents in the public transit network. The service alerts are published by transit agencies or submitted by citizens from dedicated city issue management system (i.e., City Feed [13]) or social networks.

B. *Distributed Stream-based Processing (DSP) and Distributed Messaging System (DMS)*

In order to process high-velocity real-time data, we designed an architecture which combines Distributed Stream-based Processing Engine (DSPE) and Distributed Messaging System (DMS) (Figure 3). In such architecture a Publish / Subscribe model decouples synchronized data processing from asynchronous data sending and consuming. Apache Storm [14] and RocketMQ [15] were used to implement this architecture.

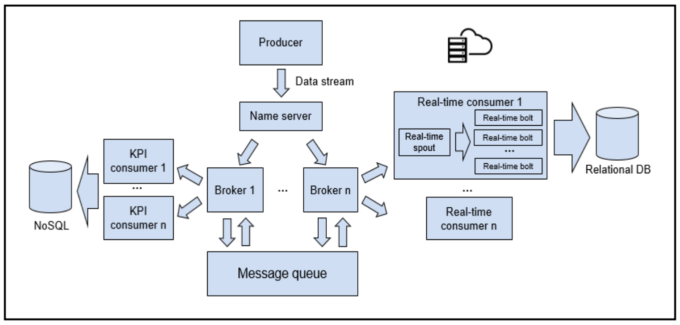


Figure 3. The DSP architecture in MOBANA

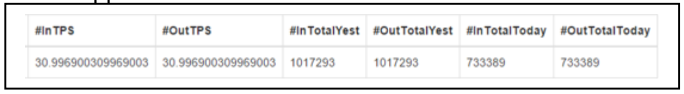
RocketMQ is an open source DMS; it includes five modules, described in Table I. We implemented a group of producers that emit real-time data stream and we used Apache Storm, an open-source DSPE, to implement three groups of data consumers. Thanks to the scalability of RocketMQ and Apache Storm, these producers and consumers can be easily deployed and managed by cloud servers. A Data Dispatch Manager (DDM) listens and receives real-time data, then it invokes the Real-time Data Producer to send messages to brokers. We defined GTFS and RT, respectively static and real-time, as data topics. GTFS brokers serves Static Data Consumer, which checks the consistency of RT data with GTFS files. If RT data are inconsistent, it will invoke GTFS Converter to check and update GTFS files.

Table I. Modules in RocketMQ

Module	Description
Name server	A stateless server cluster that registers brokers and topics. Producers and consumers get route information of master brokers which serve specific topics from name server.
Brokers	The server cluster that receives data of pre-defined topics and stores messages in "commit_log" and message queues, which are the files in which messages are stored. Brokers follow the master / slave model, which enables consumers to consume data from slave brokers when messages accumulate in a master broker.
Message queue	Group of lightweight message queues where only meta-data of messages are stored. Messages are written and consumed in sequence.
Producer Consumers	Producers send messages to brokers that serve the same topics. Consumers consume messages from brokers of topics they subscribe. RocketMQ supports fault-tolerant message consuming, when there are errors consumers will re-consume the data.

RT brokers serve two groups of consumers, KPI and RT consumers. Key Performance Indicator (KPI) Consumers incrementally compute and update in real time KPIs on a NoSQL based database, e.g., MongoDB. For each KPI, at least one dedicated consumer is deployed. Each message will be broadcasted to consumers of each KPI. RT Consumers store and update real-time transit information (e.g., delays, vehicle positions). These data are stored in a relational database, which is used for data warehousing, restoring and validation.

Data Storage Builder and Dashboard Builder are developed to re-build Data Warehouse and KPI dashboard if an unexpected crash happens.



#InTPS	#OutTPS	#InTotalYest	#OutTotalYest	#InTotalToday	#OutTotalToday
30.996900309969003	30.996900309969003	1017293	1017293	733389	733389

Figure 4. Cluster monitoring

MOBANA also supports cluster status monitoring (Figure 4), which monitors the progress of data producing and consuming, and the workload of brokers in cluster. The measure Transactions per Second (TPS) are monitored for producers and consumers. TPS shows the performance of producers and consumers, and, additionally, it can help balance the load of cluster by monitoring TPS_{in}, the average TPS of producers, and TPS_{out}, the average TPS of consumers. Let us illustrate how we use these measures to balance system performance:

(1) When $TPS_{in} > TPS_{out}$, consumers require more computing resources, and the higher Wait Time in Queue (WQ) of messages will cause a higher delay in message consuming. The possible solutions include: a) allocating more CPU cores to existing consumers, or b) deploying more consumers and brokers in server cluster; c) delete expired data, not a good solution for analytics because it may lose information.

(2) When $TPS_{in} < TPS_{out}$, which happens after resuming from unexpected crash or blocking of producers, all data in message queue will be consumed with a longer WQ. The possible solution is like in the above case.

(3) When $TPS_{in} = TPS_{out}$, consumers can process data in real-time / near real-time with a very short WQ and an almost unperceivable delay, which is the ideal status of cluster.

C. *Real-time Data Analytics*

Thanks to DSP, MOBANA supports real-time data analysis. Of course, it provides also historical data for trend analysis and reporting. Here below, we illustrate the steps of designing the real-time data analytics, by using the KPI "Delay of Vehicles" as an example.

Step 1 - KPI identification: we use HIGO [16], a stakeholder oriented business performance framework, to identify stakeholders and related KPIs. Accordingly, KPIs for public transit include: (a) Cost indicators, which assess the unit costs of input and output and the productivity and utilization rate of the resources, e.g., average travel time, average cost to customer and fare recovery ratio; (b) Quality indicators, which measure the consistency of process input and output in term of compliance, customer satisfaction and availability, e.g., stop overcrowding, urban area overcrowding, quality perceived and network condition; (c) Service indicators, which measure the time aspects of the service in term of service duration, punctuality, flexibility and fulfillment, e.g., delay at stops, path reliability, delay forecasting efficiency, land and social inclusion, average speed of vehicles. Let us consider the KPI "Delay of Vehicles", and define its measure "delay at stop" as follows:

$$D_i = \begin{cases} 0, & T_V \le T_S \\ T_V - T_S, & T_V > T_S \end{cases} \tag{1}$$

Here D_i is the delay of a vehicle at stop *i*, T_V is the actual arrival time of vehicle at stop *i*, and TS is scheduled arrival time of vehicle should at stop *i*.

Step 2 - Dimensional Fact Model (DFM): after defining KPIs, we design the conceptual model of Data Warehouse, namely a dimensional Fact Model (DFM) [17], which define fact (delay), related dimensions (date, route, and stop), and measures (average delay) as in Figure 5.

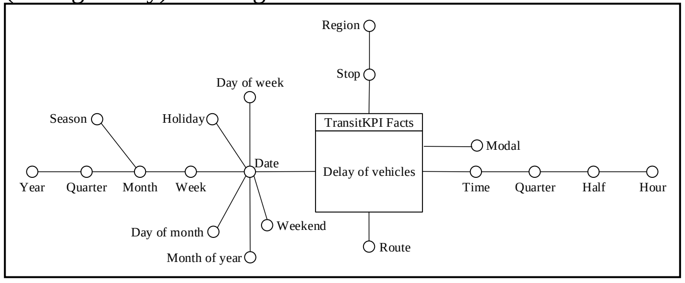


Figure 5. Dimensional Fact Model of KPI: Delay of vehicles

Step 3 - Building data warehouse: Based on DFM defined in step 2, ETL (Extract, Transform, Load) transformations are implemented. Data can be integrated in a relational database (e.g., PostgreSQL) or NoSQL database (e.g., MongoDB). Relational database is for data validation and historical data query; while NoSQL supports real-time data updating and query for transit maps and real-time KPI dashboard.

Step 4 - Incremental updating: KPI Consumers incrementally update data warehouse in MongoDB. Updated KPIs are read by the Dashboard Publisher, a message dispatch component implemented by Node.js. UI components receive the emitted messages and update the KPI dashboard. Figure 6 is an example of real-time KPI dashboard, showing the delay of vehicles for each route and stop.

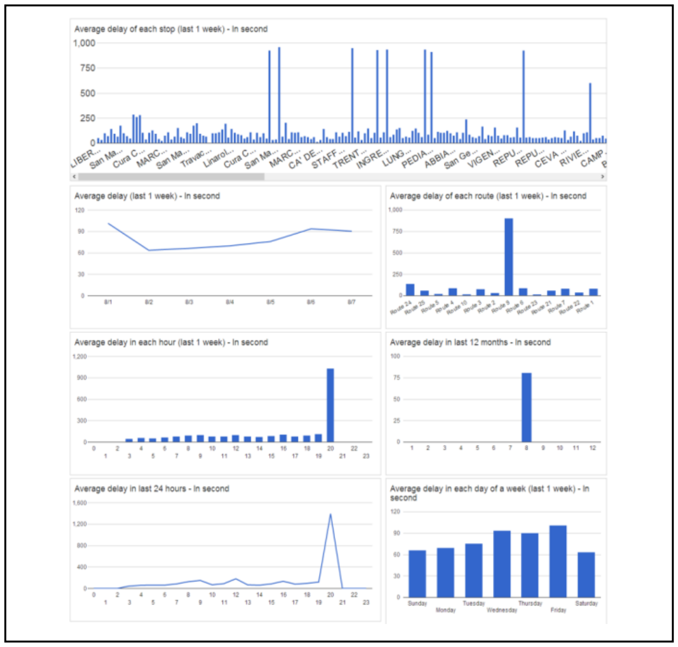


Figure 6. KPI dashboard: vehilce elay

D. *Data Visualization: Transit Map*

MOBANA displays the status of the public transit network, shows real-time vehicle positions and highlights delayed vehicles. Figure 8 shows the public transit map, which helps stakeholders to monitor the current status of public transit network by comparing scheduled and current position. Inspired by [18], we do not directly display the actual positions of vehicles by GPS data but we use a kind of position prediction, because: a) the availability of data is scarce, e.g., in Italy, up to 2015, only the city of Bari published APIs on vehicle position [19][20]; b) the volume of GPS data will drop network performance. Differently from [18], we combine static GTFS and real-time delay prediction from transit agencies, which can reduce the transmission load by filtering the real-time messages. Furthermore, part of position computation cost shifts from server to browser, which reduces the load on servers. Let us illustrate our approach.

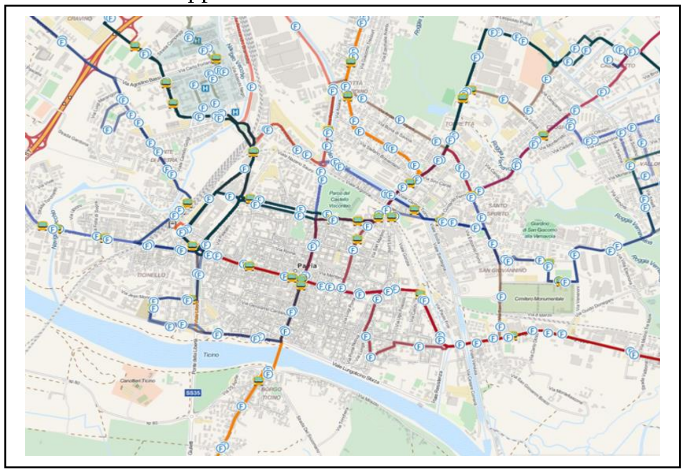


Figure 7. Public transit map of Pavia

Firstly, when the server initializes, the Map Data Generator gets the latest GTFS data loaded by GTFS Loader, and gets all the geometry information of key points in each route. Secondly, Path Normalizer, the module that normalizes the paths, computes the distances between adjacent points in each route. It divides the trip into identical segments and calculates the position of points between the segments. Finally, it builds lookup tables for trip progress (also called normalized paths) which contain progress of points with positions of each route. Algorithm 1 illustrates how the Path Normalizer calculates distances between points of a trip, and how it uses a sphere model given in [21] to locate a point (the key points pre-defined in Shape.txt by GTFS editor) onto a segment. Table II shows an example of normalized path. Obviously, the more segments of the path, the more accurate the lookup table. For instance, 1.000 calculations can make the lookup table accurate up to 0.1%. As matter of fact, we define the optimal number of segments of trip i by $s_i \geq \frac{l_i}{V}$. Here l_i is the length of trip i, V is the length of vehicle, and n_i, the amount of key points in trip *i* is $s_i + 1$. E.g., when a trip is 11 kilometers, its minimal segments are 1.000, and the length of each segment is 11 meters. If we consider that the bus length is 11 meters, results are good enough. However, for longer vehicles, e.g., trains, we should choose larger s_i.

Afterwards, the normalized paths are loaded, and scheduled positions are searched from normalized paths on client-side. Meanwhile, on server-side, Event Emitter periodically checks the updates on delays of each trip. If updates are detected, it selects corresponding trips at the current time, computes the real progress of these trips by delays, and sends the progress as events to the client-side. Finally, the client-side displays the new positions of vehicles by searching the corresponding coordinates of the progress sent by Event Emitter. The progress of a trip can be calculated by Algorithm 2. Here $delay_i$ is the delay prediction of trip *i* measured in seconds, *t* is the timestamp of now, δ is the time interval between now and the time when the next calculation will start. The positions of the vehicle in the next δ seconds will be calculated by searching points from the normalized paths. δ can be adjusted to leverage the timeliness of vehicle position prediction, network traffic and server load. For example, in rush hour we can set a higher δ (e.g., 10 seconds) in order to reduce the network traffic and let client-side compute the movements of next δ seconds. From our observation in Pavia, Italy, the maximal running trips at the same time in rush hour is 55 (8 AM). Meanwhile, the maximal real-time feeds received by DDM every second in rush hour is 150. In the worst case, the maximal updates of progress to be transmitted in the time interval δ is equal to the number of running trips. Therefore, only 55 progress pairs (new progress and trip id) in about 1,500 feeds are sent to clients every 10 seconds. Finally, the progress at each second of the next 10 seconds is calculated accordingly on client-side.

Table II. An example of normalized path for a trip

Trip progress	Location (latitude, longitude)
0%	45.2107672, 9.1630315
0.1%	45.2106254, 9.1630712
0.2%	45.2106245, 9.16300692
…	…
100%	45.2106843, 9.1630701

Algorithm 1 Normalize paths

```
**Input:** *points*
**Output:** *normalized points*
sum = 0
oldPoint = $point_0$
```

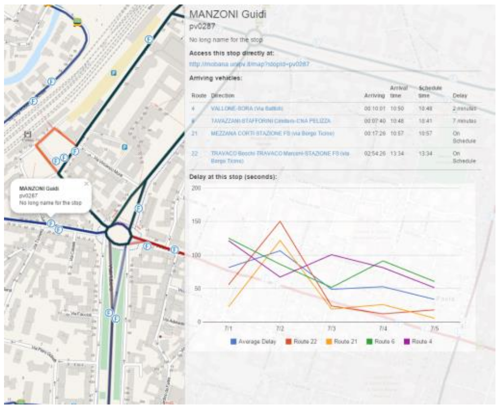
```
For all $point_i$ in $points$
    compute sphere distances $d$ between $oldPoint$ and $point_i$ //given in [21]
    $totalDistance = totalDistance + d$
    $distances.push(d)$
    $oldPoint = point_i$
End for
$K = points.size$
$D = distances.size$
For $j = 0$ to $K$
    $targetDistance = j / K * totalDistance$
    $k = 0$
    $dSum = 0$
    While $(dSum + disctances_k) < targetSum$ and $k < D$
        $dSum = dSum + distances_k$
        $k{+}{+}$
    $k{-}{-}$
    $delta = targetSum - dSum$
    compute bearing $b$ between $point_k$ and $point_{k+1}$ //given in [21]
    compute next position $p$ from $point_k$ by $b$ and $delta$ //given in [21]
    $normalizedPaths.push(p)$
End for
return $normalizedPaths$
```

Algorithm 2 Compute trip progress with delay prediction

```
Input: $delay_i$, $t$, $\delta$
Output: $nextProgress$, $currentProgress$
For all $trip_i$ in $trips$
    $nextProgress_i = currentProgess_i$
    if $startTime_i < t$ and $endTime_i > t$
        $currentProgess_i = (t - startTime_i) / (duration_i + delay_i)$
        $nextProgress_i = (t + \delta - startTime_i) / (duration_i + delay_i)$
        if $nextProgress_i > 1$
            $nextProgress_i = 1$
        if $currentProgess_i > 1$
            $currentProgess_i = 1$
End for
return $nextProgress$, $currentProgress$
```

E. Data Sharing

MOBANA provides on cloud data sharing services (i.e., Transit Analytic Service and Transit Information Service). It delivers KPI updates and reports on public transit network to public transit agencies and municipality officers. It enables citizens to query in real-time public transit information by web or smart phone, and shows routes and stops, timetables, real-time positions of buses, delay and ETA (Estimated Time of Arrival). Figure 8. shows the information panel of a stop, which displays information on routes, link of timetables, real-time delay, ETA and statistics of average delay on this stop in last five days. Furthermore, MOBANA features web services for third parties to access transit analytics and transit information.



MANZONI Guidi
pv0287
No long name for the stop
Access this stop directly at:
http://mobana.unipv.it/map?stopid=pv0287
Arriving vehicles:

Route	Direction	Arriving	Arrival time	Schedule time	Delay
4	VALLONE-BORIA (Via Battisti)	00:10:01	10:50	10:48	2 minutes
6	TAVAZZANI-STAFFORINI Cimitero-CNA PELIZZA	00:07:40	10:48	10:41	7 minutes
21	MEZZANA CORTI STAZIONE FS (via Borgo Ticino)	00:17:26	10:57	10:57	On Schedule
22	TRAVACO Bocchi TRAVACO Marconi-STAZIONE FS (via Borgo Ticino)	02:54:26	13:34	13:34	On Schedule

Delay at this stop (seconds):

Figure 8. Information panel of stops

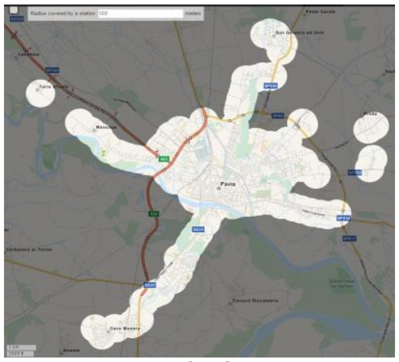
III. Case Study: Pavia Public Transit

This section explains how MOBANA was used on public transit in Pavia, Italy.

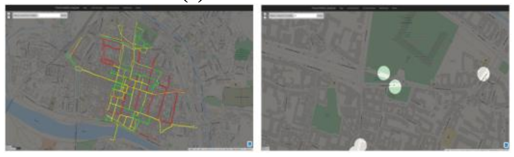
Pavia is a university city in Northern Italy with 68,000 inhabitants, where LINE, the public transit agency, runs 19 bus lines. In 2015, Mobility Analyzer was deployed as a pilot. In the following sections we describe each phase of data lifecycle.

Data gathering and conversion: We mapped GTFS files by GTFS Editor since LINE has no GTFS files. We mapped 14 bus lines (totally 30 bus routes). Real-time delay prediction were received from the Data Dispatch Monitor (DDM) service provided by LINE. Additionally, accessibility data on the urban area [22] were integrated into OSM files and loaded by Open Street Map (OSM). A Static Data Monitor (SDM) and a Real-time Data Monitor (RDM) were implemented, to receive updates on static data and real-time feeds. When changes are detected, the GTFS Converter updates GTFS files and Mobility Analyzer reloads such files. Generally, the SDM checks updates every morning before the first trip starts. RDM receives feeds of real-time delay, converts feeds to GTFS-Realtime Trip Update Feeds, and invokes data producer to emit these feeds to different data consumers.

Data processing and storage: One producer and two classes of consumers were implemented. Producer filters feeds and dispatches feeds to consumers (only the feeds of the selected 14 routes are sent to consumers). KPI Index consumers incrementally calculate and update KPI dashboard. The processed analytical data are stored in MongoDB for quick updating and searching (MongoDB is a scalable database on NoSQL and it is convenient when developing modules using Node.js.) Volumes are high: indeed, real-time feeds average 1.5 million in every workday day. Real-time Delay Consumers store the raw trip update feeds in PostgreSQL for KPI reports and data validation. It also filters the feeds by checking the records in MongoDB, only when the delay value of one trip changes, this update is sent to the presentation module. Typically, only one instance is fast enough for each producer or consumer in Pavia case. In section "Performance evaluation", the performance of the architecture in Pavia case will be illustrated.



(a) Land inclusion



(b) Social inclusion

Figure 9. Land and social inclusion

Data analysis: Three KPIs were selected as a proof of concept, namely: delay of vehicle, land and social inclusion, and accessibility of stops. These KPIs cover all stakeholders and three different data sources: real-time data, static data and accessibility data. Delay of vehicle are analyzed by multiple dimensions, i.e., stop, route, date, time, week, month, etc. A KPI dashboard, which shows real-time delay and average delay of vehicles by above dimensions, was developed for transit agencies and Pavia municipality. On the public transit map, citizens can search delays of each stop. The KPI "land inclusion" (see Figure 9 (a)) is based on stop data stemming from GTFS files. The radius of each stop which indicates the coverage of the stop was set as 500 meters. Municipality and transit agency can evaluate coverage of the whole transit network. The KPI "social inclusion", which indicates the accessibility of routes and stops, can help municipality to improve infrastructure for disabled people, let them go to bus stops with no difficulties. Figure 10 (b) shows the accessibility of transit network in central Pavia, the left figure shows the accessibility of bus routes, and the right figure shows the accessible stops that can be easily reached by wheelchair users.

Pavia transit map: The public transit map of Pavia shows the real-time position of each trip in Pavia. Users can search delay, ETA and time table of each route on this stop. Additionally, the measure "delay at stop" is also shown in the information panel when clicking a stop on the transit map (see Figure 9).

Data sharing: The KPI "delay of vehicle" is deployed as RESTful web services and can be shared by third-party applications. Another application in IRMA project called "Pavia Public Transit", used the web services published by MOBANA to provide bus information services for citizens.

IV. Performance Evaluation

In this section, we describe performance evaluation of MOBANA. The evaluation was performed in Pavia, Milan and New York with different resources, thus scalability of proposed system is also shown in testing results. These cases represents the typical data volume and TPS in different size of cities. An overview of datasets used for testing is given in Table III.

Table III. Datasets used for testing in Pavia and Milan

Case	#Stops	#Routes	#Trips	#Real-time feeds
Pavia	477	30	2,313	13,864,672
Milan	4,889	156	138,693	102,656,885
New York	18,159	1,338	206,112	527,816,841

Specifically, the original real-time feeds of vehicle delay are in JSON format, which contain timestamp, real-time delay (in second), ETA, corresponding stop, route, trip, etc. An example of a vehicle delay feed in Pavia is as follows:

```
{ "uid":"19", "eid":"19", "code":"p001", "shortdescription":"1", "description":"MONTEMAINO-SAN
MARTINO Centro (no scala)", "terminaldescription":"SAN MARTINO CENTRO", "vocalmessage":"",
"itineraryuid":"932096", "dutyuid":"0", "direction":"2", "routinguid":"74", "targetuid":"8445",
"time":21308, "arrivaltime":21308, "starttime":21308, "delay":0, "originaldelay":0, "entityuid":"0",
 "lasttargetuid":"-1", "distance":0, "theorical":1, "predictable":1, "spatialday":0, "stall":"" }
```

Typically, each feed package contains multiple feeds and the average size of a feed package sent by DDM is 4 KB.

We tested two measures, namely Wait Time in Queue (WQ) and processing time in single server node which has only 8-core CPU (AMD Opteron Processor 6238) and 8GB memories. In Pavia case, we made a five-day test, with an average data volume of 2.8 million per day. Figure 10 shows the average WQ and processing time of consumers by hour in Pavia case. Two consumers (GTFS and RT) are deployed and each consumer uses maximal 3 CPU cores. Figure 11 plots the average WQ of Pavia case with different number of CPU cores. The average WQ and processing time for each message, especially in rush hour (e.g., 7 AM, 13 PM and 17-19 PM), keeps small because: 1) the message queue decouples data consuming and producing, thus, the average processing time of consumers is not affected; 2) computing resources for data consumers are automatically balanced when data accumulation in message queue increases. In test case of AVG WQ-3 in Figure 11, the minimal number of CPU cores is set to 1 and the maximal number is set to 3, while in test case AVG WQ-1, the number is set to 1. From the result we can see that a low configuration server is fast enough for a small-size city.

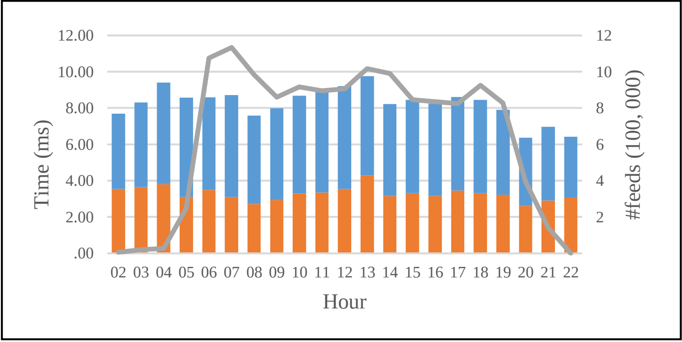


Figure 10. Performance of single server node in Pavia case

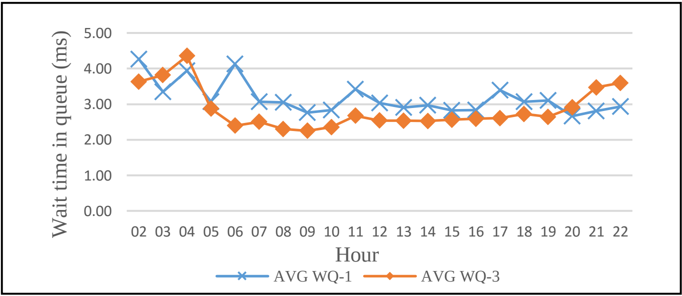


Figure 11. WQ with different number of CPU cores in Pavia case

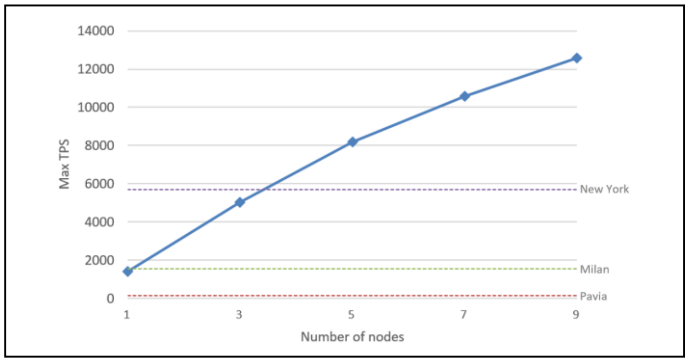


Figure 12. Throughput of MOBANA with increasing number of server nodes vs. maximal TPS of real-time public transit feeds in Pavia, Milan and New York

Finally, in order to test the performance, scalability and reusability of our system in larger cities, we did a test using GTFS files of Milan public transit and New York city transit, and simulated real-time data. Since we didn't get real-time data from public transit agencies in Milan and New York, we estimated the data volume of rush hour instead, based on the maximal TPS_{in} (i.e., 150) for 477 stops in Pavia case, i.e., TPS_{in} is equivalent to 1,537 for 4,489 stops in Milan, and 5,710 for

18,159 stops in New York. We increased the simulated feed stream and tested the maximal TPS with increasing number of server node. The test result is shown in Figure 12. From this test we got the minimal computing resources we need, only a three-machine low configuration cluster is fast enough to process real-time public transit feeds for a big-size city in Europe. The result also shows a good scalability.

V. Related Works

In this section we summarize researches and systems for public transit performance monitoring that use open transit data. We reviewed not only transit analytic systems that serve transit agencies and municipalities, but also traveler information systems that provide analytical results to citizens. In Table V, we compare our system with similar solutions and systems.

A. Analytic systems on public transit data

As GTFS data become more available, new approaches for measuring level of service of public transit in an aggregate way based on GTFS data are used in many systems [23]. In [24], authors use GTFS and basic population count at group block level to measure Transit Opportunity Index (TOI). Besides GTFS, [25] uses Service Interface for Real-time Information (SIRI) format data to analyze frequent delays, journey time spending, traffic signal waiting times and link travel times IQR (Interquartile Range) variations. [26] analyses ridership in New York public transit network using GTFS, vehicle location data and fare data. [27] proposed a visual analytic system for intelligent transportation in metropolitan, which integrates large-scale GPS data. However, all mentioned solutions do not support online analytics and visualization, even if real-time data are used. [28] proposed a multi-scale/multi-resolution design of the ITS information infrastructure and the available options at each distributed storage level. It integrates GTFS, social network data and sensor data to evaluate Service Level Agreement (SLA) of urban transit networks. However, even though it provides an architecture which processes high volume of data, analytics are not in real-time (based on ETL and data warehouse). [8] analyzes KPIs selected from the Transit Capacity and Quality of Service Manual (TCQSM), e.g., average headway, hours of service, percentage of transit-supportive areas covered. However, it takes more than 6 hours to get the analytic results. [29] presents a smart timetable services for travelers which uses crowd sensing data and GTFS to estimate Estimated Time of Arrival (ETA). Compared to the GPS data or other public transit data, the coverage of crowd-sensing data is low, because users need to install the app and open the mobile it when on vehicles. STAR-CITY integrates heterogeneous transportation data using semantic web. It provides spatio-temporal analysis and diagnosis of traffic status, and performs time-series prediction of travel times. However, the diagnosis and prediction are time-consuming, and the results are periodically updated. In [31] authors use IBM InfoSphere Streams for scalable and real-time intelligent transportation services. The proposed platform has good performance and scalability in processing GPS data. However, it only targeting GPS data, and the data streams are not exchangeable format for other system / services.

B. Real-time vehicle position visualization

Some authors proposed vehicle position visualization based on GTFS data. In [30], a real-time vehicle movement visualization which uses GTFS, transit feeds of delay and GPS data are proposed. It processes data in multi-layer spatio-temporal grids, and it periodically updates vehicle positions for an efficient and smooth vehicle movement visualization. However, it doesn't provide further analysis on public transit data to users. [18] is a real-time public transportation visualization framework for Ulm, Germany. It is the first framework who simulates vehicle positions in real-time based on GTFS data. However, it doesn't support real-time data

Table V. Comparison of existing solutions / systems

Solutions	Paradigm	Data soruces			Analytics
		Static	Real-time	Other data & format	
[24]	Parallel batch processing	GTFS	-	Basic population count by block	Transit Opportunity Index (TOI)
[25]	Batch processing	GTFS	SIRI	OSM files	Frequent delays, journey time spending, traffic signal waiting times before and after public transportation priorities, etc.
[26]	Batch processing	GTFS	Shape file streams	Boarding location data from Automated Vehicle location (AVL) system	Ridership reports
[27]	Incremental updating	-	-	Taxi GPS data	Statisitcs of taxis: e.g., trajactory, speed
[28]	Stream processing + ETL	GTFS	-	GPS, social media, sensor data	SLA
[29]	Crowd sensing	GTFS	-	Extensible Messaging and Presence Protocol (XMPP)	Crowdedness information and ETA
[8]	Batch processing	GTFS	-	National Transit Database (NTD)	Average headway, hours of service, percentage of transit-supportive areas covered, etc.
[30]	Periodic updates	GTFS	GTFS-Realtime	GPS data	-
[11]	Semantic web	-	-	Bus stream, social events, weather, etc.	Spatio-temporal analysis and diagnosis of traffic status, prediction of travel time
[31]	IBM InfoSphere Streams	-	-	GPS data	Traffic statistics, travel times and shortest path
MOBANA	DSP (Apache Storm) + DMS (RocketMQ)	GTFS	GTFS-Realtime	OSM files based on project "Pavia Accessibile"	Vehicle delay, ETA, land inclusion, social inclusion (Pavia case)

VI. Conclusions and Futurework

We have illustrated a distributed stream-based information system for analyzing performance of public transit, called MOBANA (MOBility ANAlyzer). MOBANA integrates heterogeneous data sources and, hence, can serve a wide range of stakeholders and cover any scheduled transit (bus, metro, and tramway). Compared to similar systems, it integrates more data sources into a standard exchangeable data format, i.e., GTFS and GTFS-Realtime. Second, MOBANA combines static and real-time data for vehicle position, and, thus, drops the traffic of network and load on server. Third, thanks to such selective processing and to the architecture which combines DSPE and DMS, MOBANA achieves a cost-effective performance. Actually, as we show in the Pavia case study and in a preliminary test for Milan and New York, a small number of low-power servers can process the volumes of typical cities. Finally, MOBANA can be easily be adopted by a community of cities, thanks to the exchangeable data formats and the scalable and easy-to-configure architecture.

As future works, we have identified various aspects:

- Data sources: develop an ontology-based data management approach for heterogeneous data integration, to support additional data sources, e.g., crowd data and social network data. Currently, we have already developed such approach for other services (i.e., City Feed).
- Data analysis: extend KPI management, in order to support a wider analysis in dashboard and data visualization modules.
- Data quality: develop a Lambda-like architecture paralleled with DSP for data validation.

References

[1] W. Zeng, C. W. Fu, S. M. Arisona, A. Erath, and H. Qu, (2014). “Visualizing mobility of public transportation system,” IEEE Transactions on Visualization and Computer Graphics, 2014, pp. 1833-1842, 20(12).

[2] G. Motta, D. Sacco, T. Ma, L. You and K. Liu, “Personal Mobility Service System in Urban Areas: the IRMA Project,” IEEE Symposium on Service-Oriented System Engineering (SOSE), Mar, 2015, pp. 88-97.

[3] X. L. Dong and D. Srivastava, “Big data integration,” IEEE 29th International Conference on Data Engineering (ICDE), Apr, 2013, pp. 1245-1248.

[4] Z. Zheng, P. Wang, J. Liu and S. Sun (2015). “Real-time big data processing framework: challenges and solutions,” Vol.9(6), Applied Mathematics & Information Sciences, pp. 3169.

[5] GTFS: https://developers.google.com/transit/gtfs/?hl=en

[6] Transit Agencies Providing GTFS Data: http://www.gtfs-data-exchange.com/agencies

[7] GTFS-realtime:https://developers.google.com/transit/gtfs-realtime/reference?hl=en

[8] J. Wong, “Leveraging the general transit feed specification for efficient transit analysis,” Vol.2338, Transportation Research Record: Journal of the Transportation Research Board, 2013, pp.11-19.

[9] A. A. Chandio, N. Tziritas and C. Z. Xu, “Big-data processing techniques and their challenges in transport domain,” Vol. 1, ZTE Communications, 2015, pp.10.

[10] Transit Feeds: http://transitfeeds.com/search?q=gtfsrt

[11] F. Lécué, S. Tallevi-Diotallevi, J. Hayes, R. Tucker, V. Bicer, M. L. Sbodio and P. Tommasi, “Star-city: semantic traffic analytics and reasoning for city,” Feb, 2014, Proc. of the 19th international conference on Intelligent User Interfaces, ACM, pp. 179-188.

[12] G. Motta, L. You, D. Sacco, T. Ma and G. Miceli “Mobility Service Systems: Guidelines for a possible paradigm and a case study,” Oct. 2014, IEEE International Conference on Service Operations and Logistics, and Informatics (SOLI), pp. 48-53.

[13] G. Motta, L. You, D. Sacco and T. Ma, Apr. 2014, “City feed: A crowdsourcing system for city governance,” IEEE 8th International Symposium on Service Oriented System Engineering (SOSE), pp. 439-445.

[14] Apache Storm: http://storm.apache.org/

[15] RocketMQ: https://github.com/alibaba/RocketMQ

[16] A. Longo and G. Motta, “Design processes for sustainable performances: a model and a method,” Sep. 2005, Business Process Management Workshops, Springer Berlin Heidelberg, pp. 399-407.

[17] M. Golfarelli, D. Maio and S. Rizzi, “The dimensional fact model: A conceptual model for data warehouses,” Vol.7, 1998, International Journal of Cooperative Information Systems, pp.215-247.

[18] UlmApi/livemap: https://github.com/UlmApi/livemap

[19] Public Feeds WIKI:
https://code.google.com/archive/p/googletransitdatafeed/wikis/PublicFeeds.wiki

[20] Bari Bus Support Service API: http://bari.opendata.planetek.it/OrariBus/v2.1/

[21] Calculate distance, bearing and more between Latitude/Longitude points: http://www.movable-type.co.uk/scripts/latlong.html

[22] A. Greco and G. Valentina, “Accessibility and Social Sustainability: Assessment tools for urban spaces and buildings,” 2013, 29th Conference on Sustainable Architecture for a Renewable Future. Munich.

[23] M. Nazem, M. Trépanier and C. Morency, “Revisiting the destination ranking procedure in development of an Intervening Opportunities Model for public transit trip distribution,” Vol. 17, 2015, Journal of Geographical Systems, pp. 61-81.

[24] K. Bertolaccini and N. E. Lownes, “Using GTFS Data to Measure and Map Transit Accessibility,” 2015, Transportation Research Board 94th Annual Meeting (No. 15-6045).

[25] P. Syrjärinne, J. Nummenmaa, P. Thanisch, R. Kerminen and E. Hakulinen, “Analysing traffic fluency from bus data,” Vol.9, 2015, Intelligent Transport Systems, IET, pp. 566-572.

[26] B. Suchkov, M. Boguslavsky and A. Reddy, “Development of a New, Lightweight GTFS Real Time Stringlines Tool to Visualize Subway Operations and Manage Service at New York City Transit,” 2015, Transportation Research Board 94th Annual Meeting (No. 15-4665).

[27] S. Liu, J. Pu, Q. Luo, H. Qu, L. M. Ni and R. Krishnan, “Vait: A visual analytics system for metropolitan transportation,” Vol.14, 2013, IEEE Transactions on Intelligent Transportation Systems, pp. 1586-1596.

[28] K. Lantz, S. Khan, L. B. Ngo, M. Chowdhury, S. Donaher and A. Apon, “Potientials of online media and location-based big data for urban transit networks in developing countries,” 2015, Transportation Research Board 94th Annual Meeting (No. 15-4942).

[29] K. Farkas, “Smart Timetable Service Based on Crowdsensed Data,” European handbook of crowdsourced geographic information. Ubiquity Press, pp. 1-13, in press.

[30] H. Bast, P. Brosi, S.Storandt, “Real-time movement visualization of public transit data,” 2014, Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems. ACM, pp.331-340.

[31] A. Biem, E. Bouillet, H. Feng, A.Ranganathan, A.Riabov, O.Verscheure, H.Koutsopoulos and C. Moran. “IBM infosphere streams for scalable, real-time, intelligent transportation services,” 2010, Proceedings of the 2010 ACM SIGMOD International Conference on Management of data, ACM, pp. 1093-1104.