

Aalto University
School of Science
Master's Programme in ICT Innovation

Marco Zugliani

Serverless data processing applied to big mobility data

Leveraging activity recognition data from smart-phones to estimate CO₂ emissions in everyday transportation

Master's Thesis
Espoo, September 17, 2018

Supervisor: Professor Petri Vuorimaa, Aalto University
Advisor: Julien Mineraud, Postdoc, University of Helsinki

Aalto University
 School of Science
 Master's Programme in ICT Innovation

ABSTRACT OF
 MASTER'S THESIS

Author:	Marco Zugliani		
Title:	Serverless data processing applied to big mobility data. Leveraging activity recognition data from smartphones to estimate CO ₂ emissions in everyday transportation.		
Date:	September 17, 2018	Pages:	v + 61
Major:	Digital Media Technology	Code:	SCI3023
Supervisor:	Professor Petri Vuorimaa		
Advisor:	Julien Mineraud		
<p>Personal mobility has become a relevant aspect of daily life in the modern society. The knowledge of how people move in the territory plays a central role in how cities develop as an ecosystem. Citizens gain insight into their everyday movements and can act in order to improve their lifestyle.</p> <p>This thesis describes an automated system capable of discovering daily trips and personal carbon footprint with the most common transport means, including public transports.</p> <p>Several studies have been performed on the topic of transport mode recognition. While agreeing on the use of smartphones to collect data, these studies vary mainly in the number of modalities recognized and in the sensors used.</p> <p>This work presents a novel approach that exploits an existing activity recognition technique and the serverless technology as a means to process and enrich the data with more information. This approach greatly affects the accuracy of the off-line recognition system by adding GPS and public transport information.</p>			
Keywords:	Mobility recognition, Serverless, MapReduce, Smartphone sensors, Carbon footprint		
Language:	English		

Acknowledgements

I would first like to thank my thesis supervisor professor Petri Vuorimaa of the Department of Computer Science at Aalto University for his unobtrusive yet fundamental support for my Master's Thesis.

I would like to express my gratitude to my advisor Julien Mineraud for the useful suggestions, guidance and engagement through the learning process of this Master's Thesis.

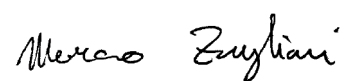
I would also like to thank Petri Martikainen, from Moprim, for giving me the possibility to work on this project.

Special thanks to my family for giving me constant support and continuous encouragement during my years of study. This accomplishment would not have been possible without them.

Finally, I would like to thank my sweet Qiu Shuang for supporting and bearing me during this time.

Thank you.

Espoo, September 17, 2018

A handwritten signature in black ink, reading "Marco Zuyliani". The signature is written in a cursive, flowing style.

Contents

Acknowledgements	iii
1 Introduction	1
2 State-of-the-art	5
2.1 Accelerometer based techniques	8
2.2 Inertial measurement units	10
2.3 GPS based	11
2.4 Hybrid techniques	12
2.5 Other approaches	14
2.6 Considerations and remarks	16
3 Data Processing	17
3.1 Mobile computing	17
3.2 Server-based processing	18
3.3 Serverless processing	18
3.4 Big data processing	21
3.4.1 MapReduce	21
3.4.2 Serverless MapReduce	22
4 Environment	24
5 Methods	26
5.1 Mobile app	26

5.2	Serverless data processing	27
6	Implementation	28
6.1	System overview	28
6.2	Data gathering	30
6.2.1	Data types	30
6.2.2	Data challenges	31
6.3	Data processing pipeline	32
6.3.1	On-demand processing	33
6.3.2	Data processing algorithm	33
6.4	Data analytics	43
7	Evaluation	45
7.1	Cost analysis	45
7.2	Accuracy	48
8	Discussion	52
8.1	Transport mode recognition	52
8.2	Cloud implementation	53
9	Conclusions	55

Chapter 1

Introduction

Mobility data has gained value in the recent years, as smart cities are seeking to improve and create new services through artificial intelligence and digitization. The knowledge of this data enables several novel applications related to smart mobility, such as improving the efficiency of cities and the quality of life of its citizens. People gain insight into their everyday mobility; cities can understand how to improve their public transportation network.

Recent studies show that 24% of the of global CO₂ emissions from fuel combustion in 2015 derives from transportation [2]. Moreover, 61% of the world transportation energy consumption in 2012 originates from personal mobility-related fuel consumption [1].

Smart mobility services require reliable data sources. This thesis contributes to Lahti's CitiCAP project, an EU-funded project that aspires to promote sustainable mobility as opposed to private transportation in order to reduce transport-related emissions. The aim of this study is, therefore, to build a reliable solution that can harvest mobility data from citizens at scale for big cities, allowing both users and administrations to make conscious mobility choices and consequently reduce CO₂ emissions related to personal mobility. Smartphones are utilized as an information source, thanks to their ubiquity. Modern smartphones embed very sophis-

ticated sensors, such as accelerometers, gyroscopes, and Global Positioning System (GPS). Recent pieces of research show that these sensors can achieve notable accuracy in activity recognition. Although they offer limited precision and sampling rate, they are capable of capturing effective information about the movement patterns.

Recent papers present different approaches to activity recognition, using various combinations of the sensors available. The majority of the works disregard GPS data and public transport information. Motion sensors are generally the preferred method for activity recognition. They achieve good discrimination accuracy, but suffer from noisy measurements (e.g., mobile phones may be handled by users during a trip). GPS-based solutions suffer from high power consumption, which limits the usability of the solution. Users could decide to abandon such a solution because of this reason. Moreover, GPS signal can be unavailable in certain conditions (e.g., during metro rides), which limits the recognition ability. Hybrid solutions promise to mitigate the accuracy problems of these two different approaches, although still suffering from high power consumption. In our case, GPS data is required for mapping the trips and for estimating the CO₂ emissions.

Collection of mobility data at city scale required considerations on scalability. The novel serverless technology is explored as a means to carry out computations. This technology allows effortless and cost-effective deployment of functionality in the cloud by providing an abstraction on the infrastructure. It has gained the attention of many cloud developers due to its flexibility and scalability and has proven as an effective environment for rapid development.

Compatibly with Lahti's CitiCAP project, the system requires high accuracy, specifically in the discrimination of motorized modalities such as car, bus, and metro.

The challenge considered in this thesis is to improve the reliability of the recognized transport modes and to limit the fragmentation of the out-

put data, by combining negligible segments. Furthermore, the adoption of a serverless architecture addresses the need for scalability.

This study builds upon an existing transport mode recognition system [13] based on accelerometers sensors, by adding GPS and public transport timetable information.

GPS data is required to extract a sound estimation of CO₂ emissions. Furthermore, it allows a substantial improvement in recognition performance and reliability. In addition, GPS information is used to limit the fragmentation of the output, so that only relevant trips are detected and reported.

Public transport information is retrieved from a Geographic Information System (GIS) with information on the local transport lines. This data enables verification of correct public transport trips, further improving the system reliability.

The system described in this thesis takes advantage of these two information sources, as well as an existing activity recognition engine, to produce a sound estimate of the carbon footprint related to personal mobility.

This document answers the following questions:

- Can GPS and public transport information improve the classification accuracy for transport mode recognition?
- How much do these two information sources affect such classification?
- Is serverless suitable for large-scale data processing?
- What are its main benefits and drawbacks?

Chapter 2 introduces the problem of transport mode recognition and discusses previous works and frameworks that aim to solve this problem. Chapter 3 describes different architectures for data processing with emphasis on the novel serverless platform. Chapter 4 outlines the environment around the study including some broader considerations. Chapter

5 highlights the main components of the platform and their objectives. Chapter 6 delves into the system components related to the novel contribution of this work. It includes a thorough description of the data processing algorithm, as well as considerations on the data limitations. Chapter 7 evaluates the performance of the system, particularly focusing on the application for the output information. Chapter 8 identifies future challenges and possible improvements of this work. Finally, chapter 9 highlights the most important findings of this work.

Chapter 2

State-of-the-art

Transport mode recognition is a problem that has been the focus of many research papers in the last decade. The earlier methods used to collect travel and commuting trips required manual work. Paper diaries, phone interviews, and web questionnaires were the most prominent sources of this information. These methods, however, showed large disadvantages, such as low response-rate and under-report of the trips [18]. Automatic methods were developed to contrast this trend. After 2000, GPS-based techniques started to be the focus of research, although they required specialized hardware, often vehicle-mounted or difficult to carry around [8, 32]. This burden prevented automatic transport mode detection from becoming widespread and hindered research due to the need of costly, specialized hardware.

After 2010, with the popularization of mobile phones and then smartphones, an unprecedented number of people gained access to a wide variety of sensors. These enable unobtrusive monitoring of human activities since they are carried continuously by a person. As a result, the vast majority of smartphone currently includes an activity recognition framework. Furthermore, a significant number of research papers has been produced on the topic.

Research has shown great interest in activity and travel mode recog-

nition in the recent years. Many research directions have been explored. Despite this, They agree on the use of smartphones as a means to collect this information. Researchers proposed a wide range of implementations, exploiting various combinations of sensors. They can be grouped into categories. The following sections explore such categories.

In their survey, Shoaib et al. [30] review some of the most relevant works prior to 2015 that focus on online activity recognition, i.e., when the classification takes place on the device that collects sensor data. They identify a total of 30 relevant studies that employ one or more machine learning classifiers to discriminate among activities. They report that most of these works utilize decision tree classifiers to categorize the data. Other popular classifiers include Support Vector Machines (SVM), K-nearest neighbors (KNN) and Naive Bayes classifiers (NBC). Most works used an offline training process in conjunction with the online classification. In these works, the authors train a model offline using a dataset collected during development. The model is subsequently deployed on mobile devices as a general model for classification. The authors observe that such approach proves to be less costly, but lacks personalization: the model may not fit every user and perform poorly in particular situations. The most used platform for the experimental development of these setups is Android, thanks to its open source nature and ease of programming. Earlier works use the Symbian platform as well. The vast majority of the works employ accelerometer data only since it enables a reasonable accuracy for most applications while preserving a minimal power consumption. Other sensors are used in some works and enable more sophisticated models and a general improvement in accuracy. Despite the accuracy improvement, these sensors are available only on a limited number of devices, while almost all modern devices offer accelerometers. The authors note that the comparison of different approaches is a hard task, due to the lack of a consistent benchmark framework for all of the studies. The recognized activities

	Ref	[22]	[13]	[29]	[10]	[19]	[17]	[20]	[16]	This work
Sources	Accelerometer	✓	✓	✓	✓		✓	✓	✓	✓
	Gyroscope		✓				✓		✓	
	GPS			✓		✓		✓		✓
	GIS			✓						✓
	Other						✓ ¹	✓ ²	✓ ³	
Transport Modes	walk	✓	✓	✓	✓	✓	✓	✓		✓
	run		✓	✓			✓	✓		✓
	bike	✓	✓	✓	✓	✓	✓	✓		✓
	car	✓	✓	✓	✓	✓			✓	✓
	bus	✓	✓	✓		✓	✓		✓	✓
	road		✓					✓		✓
	metro	✓	✓		✓	✓				✓
	train	✓	✓	✓		✓				✓
	tram		✓			✓				✓
	rail		✓					✓		✓
	plane							✓		

Table 2.1: A summary comparison of the reviewed approaches to transport mode classification.

range from basic movements (e.g., walking, standing, driving) to more sophisticated activities (e.g., using the elevator, vacuuming, brushing teeth or ironing). While these works have been fundamental to the rise of activity recognition techniques and implementations, they neglect specific transport modalities. Recently, a number of research papers have been produced on the topic of transportation identification using smartphone sensors. Table 2.1 summarizes the main approaches reviewed, comparing the sensors used for the classification and the number of classes that they can discern.

¹Rotation Vector Sensor

²Magnetometer

³Google Activity Recognition

2.1 Accelerometer based techniques

Accelerometers are sensors available in almost all modern mobile devices, besides smartphones, tablets and recent laptops offer those sensors as well. They sense acceleration forces, such as gravity or movements of the device. They have been incorporated primarily for the scope of rotating the screen orientation in such a way that content is displayed upright. They are usually tri-axial, i.e., they detect acceleration along three different axes. Activities are recognized analyzing the different patterns recorded by the sensor.

Manzoni et al. [22] present one of the earliest works on transport mode recognition using mobile phones. Their approach is based on a machine learning classifier, namely decision trees. The classifier could discriminate between 8 classes, including bus, metro, train, and car with 82% accuracy using the accelerometer data exclusively. In order to account for orientation changes, they utilize Fast Fourier Transform (FFT) coefficients computed on the total acceleration, which is orientation-invariant. This approach results in a good performance while preserving power consumption thanks to the omission of GPS data. The authors note however that, in order to estimate the CO₂ emissions, GPS data is necessary to estimate the traveled distance. Furthermore, the paper omits the number of test subjects and the duration of the data collection.

Ferrer and Ruiz [10] compare the performance of a number of machine learning classifiers trained on accelerometer data. In particular, the recurrent neural network (RNN) model achieves over 93% of accuracy. They rely on orientation-independent statistics for the classification, such as mean, standard deviation and maximum of the magnitude of the acceleration. The model has been trained to classify different means of transportation, however, they limit the motorized modalities to car and metro.

In order to exploit more advanced classifications features, orientation estimation is fundamental. Some works have explored gyroscopes as a

data source for this information, which will be presented in section 2.2.

The work of Hemminki et al. [13], which is at the base of the system described in this document, attempts to solve this estimation exclusively with accelerometers. Their approach to smartphone sensor data processing is composed of two main components: the estimation of the gravity component for the reliable computation of the linear acceleration and the definition of new features to accurately extract the mode of transportation of the user.

Gravity is estimated over a short window (1.2s), so as to retain responsiveness to changes. The signal analysis identifies low variation periods, which allow to accurately estimate the gravity component. The change in orientation of the device is handled with a reinitialization of the gravity estimation. A substantial change in acceleration direction triggers the gravity reinitialization. The reinitialization ensures that the change in orientation is not interpreted as a change in acceleration.

The reliable estimation of gravity enables to remove the gravity component from the sensor data. This, in turn, allows isolating more reliable features to estimate the transport modalities.

The authors focus on the recognition of different locomotion types, including pedestrian modalities (e.g., walking, running), non-motorized transportation (e.g., bicycle, roller skates) and motorized transportation (e.g., car, metro, bus, train).

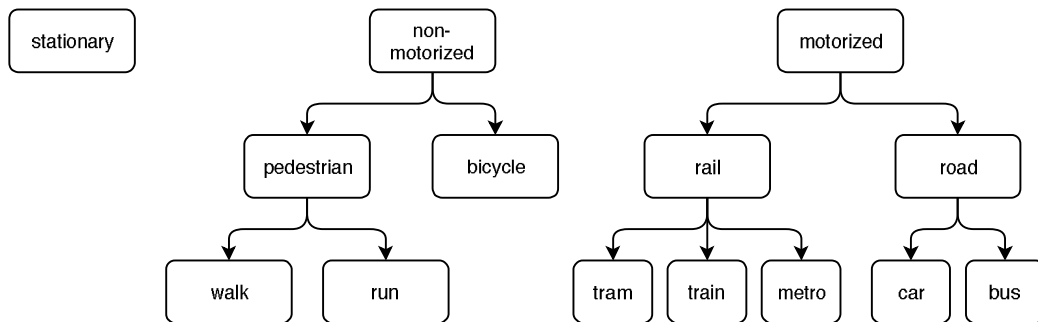


Figure 2.1: Hierarchy of classifications. Eight leaf modalities can be identified.

The classification scheme is hierarchical, as illustrated in Figure 2.1. When the confidence of the algorithm is too low to discriminate between leaf classes, the output contains one of the group classes. They report an average of over 80% for precision and recall of their solution, which is lower than the previously analyzed solutions, however, it considers a larger number of classes. Furthermore, the reliable gravity estimation allows for reliable classification in different scenarios related to the position of the smartphone on the person, such as pocket or backpack.

2.2 Inertial measurement units

Inertial Measurement Units (IMU) include accelerometers and gyroscopes. Gyroscopes represent another popular sensor used in activity recognition. They sense the change in orientation of the device and are often used in conjunction with accelerometers. Few smartphones, however, offer this sensor: Only 34% of Android and iOS smartphones released after 2015 offer gyroscopes⁴.

Heydary et al. [16] developed a machine learning classification algorithm based on a random forest to discriminate between car and bus trips using sensor data from the smartphone. They employ accelerometer, gyroscope and gravity sensor (a composite sensor relying on accelerometer and gyroscope). In order to minimize the amount of data analyzed, they filter the data based on Google Activity Recognition framework to retain only vehicular trips. They report an accuracy of 93% on over 15 hours of data, evaluated with a custom Android app running on Google Pixel smartphones. Although the discrimination of car and bus rides is intrinsically hard, due to the vehicle similarity, they are able to provide high discrimination accuracy. However, their work is restricted to these two classes and the evaluation data is limited.

⁴657/1918 models offer a gyroscope. <https://www.gsmarena.com/search.php3>.

Hemminki et al. [14] added gyroscope data to their previous gravity estimation [13]. The introduction of gyroscope data allows an improved estimation of the gravity component. The component is estimated based on the stability score, which encodes the variability of the measurements around a key-point. Accelerometer and gyroscope data are compared to find inconsistencies and prune inaccurate estimations. The final algorithm extracts vertical, lateral and longitudinal components of the motion. The vertical component is computed by subtracting the gravity component from the z-axis of the data. Lateral and longitudinal component are estimated through key periods corresponding to acceleration, braking, and turning phases. This approach improves the state-of-the-art especially in presence of high noise or sustained acceleration. Furthermore, the window size for the estimation is considerably shorter than previous solutions. This allows for more reactive updates while maintaining a high estimation accuracy. The authors, however, omit the evaluation of transport mode estimation.

2.3 GPS based

GPS is a very popular sensor used in activity recognition applications. It is frequently used along with other sensors, however, it has been utilized as the sole information source in certain works. It enables distance and speed measurements but lacks lower level context awareness. Moreover, it performs poorly in urban jungles or indoor settings where the GPS signal can be lost. In addition, it suffers from power consumption issues, leading to fast battery depletion. Many works tend to omit this sensor due to this reason, however, specific applications, such as CO₂ emission calculation, require an estimation of the traveled distance.

Kugler et al. [19] developed a system capable of estimating CO₂ emissions related to personal mobility. They employ only the GPS sensor and are able to discriminate among several mobility modes, including car, bus,

tram, metro, and train. They report almost 88% accuracy on their test set that comprised over 50 participants.

2.4 Hybrid techniques

Hybrid solutions can be developed employing combinations of sensors. The most used combination uses GPS in conjunction with accelerometers or IMUs. Geographic Information Systems (GIS) are used as well to validate public transport trips or to infer the semantic context of an activity (e.g., to discriminate restaurant from work).

Shah et al. [29] describe a decision tree classifier to distinguish different transportation modes. They consider both above-ground and underground modalities. They, however, report the performance only of the discrimination between car, bus, and train, which is slightly over 80%, using accelerometers only and over 90% adding GPS information. Their classifier employs accelerometer as well as GPS data. Furthermore, they exploit GIS to validate the public transport routes.

Lorintiu and Vassilev [20] propose a system for the computation of CO₂ emissions exploiting accelerometer, magnetometer and GPS data. They report accuracy as high as 94% without GPS and 96% with GPS. They are able to discern among seven classes, including road, rail, and plane, not distinguishing further into these classes. Moreover, they employ magnetometers, which are available on a limited set of devices⁵.

The work from Rinne et al. [27] is in many ways similar to that of this paper. They describe a system capable of discovering public transport trips in the Helsinki area. They developed the system using Google Activity Recognition APIs, GPS data and GIS information, including both static timetables and live positions of the vehicles. In the analyzed area, public transport GIS information is available through a web API from the

⁵ 54% (1033/1918) of Android and iOS smartphone released after 2015 offer magnetometers. <https://www.gsmarena.com/search.php3>.

local transportation authority as a public API⁶, similarly to our implementation. They developed a mobile app that collects and sends to a server activity recognition results and location data. The server then processes those data, in conjunction with the GIS information.

Activity recognition data is filtered to remove outliers. Common problems observed include: trips on rails are periodically recognized as being still periods triggering a transition to sleep. After sleep periods, the time to resume an active state is as long as 200 seconds, causing low responsiveness to activity changes.

Location data is collected through Android fused location provider. As in our case, the authors note that the retrieved location may occasionally fluctuate between distant positions. This is due to the different underlying positioning systems used by the provider.

Live GIS information is requested every 30 seconds from the API and stored in a database. When the data from users' devices is available, the stored public transport information is analyzed and candidate matches are selected. Although live locations provide benefits such as adaptation to the traffic conditions, they suffer from some limitations, which include:

- Coverage: only a limited subset of transportation lines are available through the live positioning system.
- Storage: despite the low number of lines available, the live API produces a large volume of data. Moreover, fresh data is required since traffic varies continuously.
- Performance: the number of locations used to compute the matches had to be limited in order to allow reasonable performance. However, reducing the location samples, the accuracy is reduced as well. The authors claim good accuracy on slower vehicles, while higher speeds produce false negatives.

⁶<https://digitransit.fi/en/developers/apis/>

- Location accuracy: both location systems in public transports and in user devices may include inaccurate points.
- Clock differences: user devices and vehicles may have clock offsets.

Static timetables are used to match trips, which are absent in the live GIS APIs. False negatives can arise with inaccurate location points and activity labels, clock differences. False positives can be observed when a user is driving close to a public transport vehicle. In order to achieve the match, they allow some distance and time tolerance. This is done to take into account the inaccuracies of the activity recognition and the location service on the device. Furthermore, locations sequences are matched with fixed position sampling. The matching requires a regular sampling rate from the user device, which is sometimes unattainable, e.g., in metro trips. Moreover, quick transfers between different vehicles may be disregarded by the activity recognition service, leading to an incorrect false negative.

Overall, they report 60% of matches for buses and trams and 43-44% for trains and subways.

2.5 Other approaches

Other sensor combinations have been explored. Although they produce improved recognition performance, they are relatively limited in adoption. This is due to the modest number of mobile devices offering these sensors. It is, however, worth mentioning few notable pieces of research.

Jahangiri and Rakha [17] employ accelerometer, gyroscope, and Rotation Vector Sensor (RVS) data in their classifier. They test various classifier models, among which random forest, bagging, and SVM offer the best performance. The models were trained to classify the basic transportation modes including car and bus. These two classes, in particular, are the most confused, given their intrinsic similarity. Nevertheless, the use of various sensors as data source allows to achieve over 94% overall accuracy (for the SVM model).

Read et al. [26] describe a system based on data fusion techniques. They focus on data cleaning and pre-processing, rather than classifying. They collect data from several sensors, including accelerometers, gyroscopes, and GPS. They note that each of these sensors produces observations with different rates and therefore need a considerable effort to align them to a consistent series of observations. In addition, they consider the case of multi-labeled trips (e.g., walking while on an escalator).

Mobile APIs

Activity recognition is a function that the major mobile operating systems provide to the developers. Both Google and Apple have frameworks that enable developers to gain easy access to the activity context of the smartphone. The two providers offer a comparable set of activities: they are able to recognize basic activities, such as walking, cycling, on a vehicle and being stationary. Furthermore, both provide annotation of the recognized activity with a confidence score.

Apple provides a single API: `CMMotionActivity`⁷. It is part of their Core Motion framework. It provides updates when an activity transition occurs (i.e., when an activity starts or ends), often sufficient for most applications.

Google provides two different APIs: `ActivityRecognition`⁸ and `ActivityTransition`⁹. Both are part of their location package and they employ only low power sensors to reduce power consumption. The former has been part of the Android ecosystem for a long time. It provides continuous notifications about the current activity the smartphone recognizes and offers some degree of battery optimization by reducing the sensing

⁷<https://developer.apple.com/documentation/coremotion/cmmotionactivity>

⁸<https://developers.google.com/android/reference/com/google/android/gms/location/ActivityRecognition>

⁹<https://developers.google.com/android/reference/com/google/android/gms/location/ActivityTransition>

interval when the motion is not significant. The latter has been recently introduced¹⁰ to ease the burden on the developer by preprocessing the activity notifications. It works similarly to Apple's CMMotionActivity API, providing only the activity changes. The new API promises to reduce battery consumption and add new activities such as differentiating between road and rail vehicles in the near future.

2.6 Considerations and remarks

Many different approaches have been examined to accomplish transport mode recognition. The main reason being that different applications have different requirements. In some contexts, GPS adds little relevant information, whereas, in others, reliable distance and speed estimation is required. The latter case holds for this research since CO₂ estimation is based on the traveled distance. Furthermore, sensor availability is an essential factor that influences the reach of the solution on the market. While accelerometers and GPS sensors are available on the vast majority of the devices (98% of device models¹¹), other sensors are less widespread. Thus, this thesis focuses on the use of accelerometers and GPS (as well as public transport GIS information) to achieve transport modality discovery.

¹⁰<https://android-developers.googleblog.com/2018/03/activity-recognitions-new-transition.html>

¹¹ 1876/1918 of Android and iOS smartphone released after 2015 offer accelerometer and GPS. <https://www.gsmarena.com/search.php3>.

Chapter 3

Data Processing

In order to produce meaningful results related to the personal mobility and carbon footprint, the activities recognized by a framework have to be processed and cleaned.

This chapter discusses three approaches to data processing: mobile computing in Section 3.1, server-based processing in Section 3.2 and server-less processing in Section 3.3. Section 3.4 introduces a recent programming paradigm (MapReduce) tailored to big data processing.

3.1 Mobile computing

When the processing happens on the user device, the approach is categorized as mobile computing (also edge computing or edge processing). It enables progressively more complex tasks thanks to the increasing computing power embedded on these devices, smartphones in particular. This approach offers a number of advantages, such as high security and scalability. Local data processing enables increased security since the data remains on the user device. Scalability comes naturally: each new user comes with his/her dedicated resources. Limitations come from the mobile nature of devices. Portability brings resource constraints, such as storage, computational power, battery, and bandwidth. These constraints are

nevertheless less pressing as technology advances.

The vast majority of activity recognition approaches reviewed in chapter 2 fall into this category. Performing data analysis directly on the device considerably reduces the bandwidth necessary due to the verbosity of signals. As an example, one day of accelerometer data sampled at 200Hz leads to almost 300MB of data. Local processing of this data reduces network traffic and server cost.

3.2 Server-based processing

Server-based approaches are on the other side of the spectrum. High-performance devices allow executions with none of the constraints of mobile computing. Servers allow complex computations to take place and large quantities of data to be stored. They can effortlessly communicate with different services to compose several sources together and produce rich information. This approach has been implemented by Rinne et al. [27] in their platform in order to combine activity recognition data, GPS and public transport GIS information. Server-based processing suffers, however, of limited scalability due to the inherent cost of infrastructure.

3.3 Serverless processing

The serverless paradigm has recently emerged as a novel approach to deploy functions on the cloud. It is sometimes referred to as Function as a Service (FaaS) due to its modular nature. Its name derives from the abstraction that it creates between developers and the underlying infrastructure. It does still require servers to run the code, however, the developers lose the control over them. Cloud providers manage the infrastructure. They have the task of maintaining, scaling, provisioning and patching the infrastructure and its runtime. There is currently a wide range of architecture offered by the cloud providers. Figure 3.1 clarifies the differences

On-Prem	IaaS	PaaS	FaaS	SaaS
Traditional on-premises cloud	Infrastructure as a Service	Platform as a Service	Function as a Service	Software as a Service
Data	Data	Data	Data	Data
Functions	Functions	Functions	Functions	Functions
Application	Application	Application	Application	Application
Runtime	Runtime	Runtime	Runtime	Runtime
Back-end Code	Back-end Code	Back-end Code	Back-end Code	Back-end Code
Operating System	Operating System	Operating System	Operating System	Operating System
Virtualization	Virtualization	Virtualization	Virtualization	Virtualization
Server Machines	Server Machines	Server Machines	Server Machines	Server Machines
Storage	Storage	Storage	Storage	Storage
Networking	Networking	Networking	Networking	Networking

Figure 3.1: Comparison of different cloud service architectures. The grayed-out blocks are those managed by the cloud provider. In serverless (FaaS in the figure), developers only provide the functionality and the data.

among the prevailing architectural paradigms offered. Serverless has been introduced at the end of 2014 by Amazon in its annual re:Invent conference. It has constantly increased its popularity and user base. Currently, all major vendors support the FaaS paradigm (AWS¹, Google², IBM³, Microsoft⁴). The OpenLambda [15, 23] project has been proposed as an open-source platform capable of outperforming other commercial solutions.

Serverless is characterized by its functional nature: developers package and deploy small code snippets that have limited functionality. Events are at the base of the invocation model. Functions execute in response to events.

Composability stands as a key element of the serverless paradigm. In order to add rich functionality to an application, different, specialized ser-

¹<https://aws.amazon.com/lambda>

²<https://cloud.google.com/functions>

³<https://www.ibm.com/cloud/functions>

⁴<https://azure.microsoft.com/en-us/services/functions>

vices, often offered by the same cloud provider, are composed together. The rich ecosystems offered by cloud providers renders it very easy for developers to link different services and achieve complex behaviors. Although services by different providers are generally interoperable, each provider encourages the use of its services, facilitating the integration among these services. This ease of integration is seen by many as a possible threat named as vendor lock-in. It indicates the dependency that is created between the customers and the vendors.

Cost modeling for serverless functions works differently than traditional on-demand compute instances. With serverless, customers pay only for what they use, however, they pay for everything they use. As observed by Eivy [9], the economics of serverless can be misleading. Therefore, carefully estimate the number of users of the platform becomes critical. Serverless is suitable for functions performing rapid actions with low traffic that may suffer sudden spikes in traffic and are compute-bound (as opposed to I/O-bound). These qualities follow from the scalability property: serverless functions can freely scale up and down to meet precisely the number of requests needed. It may scale upwards, supporting traffic bursts, as it scales downwards, supporting the “scale to zero”, where nothing is run and paid for. However, for services that expect stable high load, serverless may prove sub-optimal. Prices per executions are generally higher than a similar PaaS solution.

Academic research has produced few publications on serverless. Most of them aim at comparing commercial implementations [23, 31] or introducing its characteristics and use cases [4, 23].

Literature mostly includes practitioners guidelines [3, 28], targeting developers, rather than researchers. This reflects the highly practical nature of serverless as well as the little information about the architecture used by cloud vendors, that prefer to maintain these facts reserved.

3.4 Big data processing

Large-scale data processing requires particular approaches to be implemented efficiently. While platforms, such as serverless, allow a high degree of scalability due to their distributed architecture, special programming patterns are necessary to harness this quality.

3.4.1 MapReduce

MapReduce is a specialized programming model that aims at improving the processing of large datasets. It has been developed by Google and its model has been published in 2008 [5]. It exploits task parallelization of small functions to achieve the processing of large quantities of data. The main building blocks are *map* functions (or *mappers*) and *reduce* functions (or *reducers*). Map functions take a series of input key-value pairs and transform it to intermediate key-value pairs (e.g., a word counter would take as input text documents and produce a word-count pair for each word. A map function would analyze a single page or document).

The input of mappers is first split into small, more processable units of information in order to maximize jobs parallelization. The intermediate output of mappers is then reshuffled to simplify the job of reducers.

Reduce functions take the intermediate output of the map functions and returns the final list of key-value pairs with the conclusive results(e.g., the word counter would take the intermediate word-count pairs and generate a single value for the whole series of documents).

Mappers and reducers are controlled by some coordinator functions. They divide the input between mappers, partition the intermediate results to reducers and aggregate the final results from reducers.

Figure 3.2 illustrates a MapReduce work-flow for a word counter application.

MapReduce is used in multi-core clusters of machines to considerably accelerate computations on large sets of data. Several libraries have been

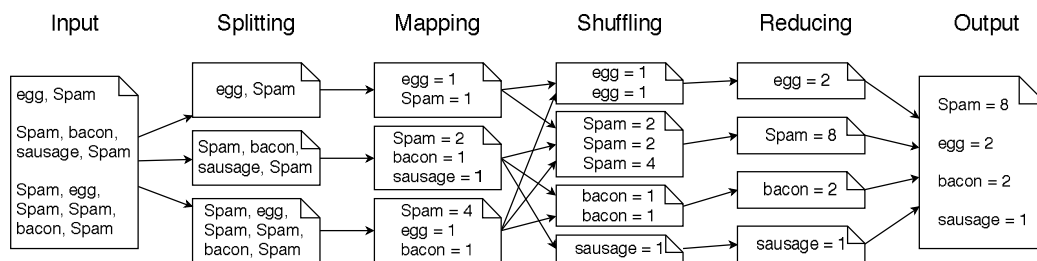


Figure 3.2: MapReduce work-flow. A word counter example using MapReduce.

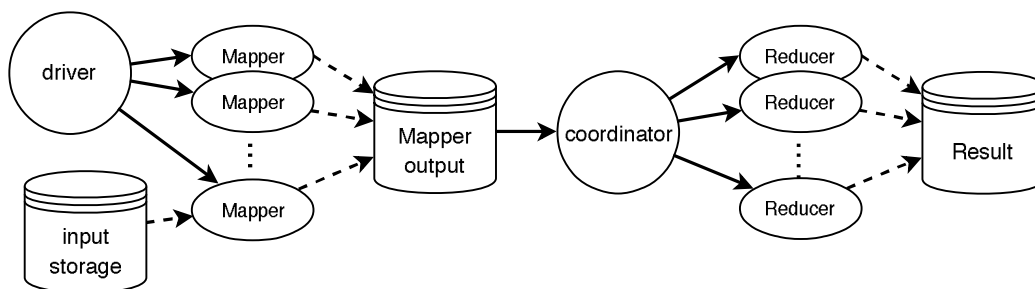


Figure 3.3: Serverless MapReduce components architecture. Circles and ovals indicate serverless functions, cylinders represent storage.

developed implementing this programming model. The most prominent are Apache Hadoop⁵ and Apache Spark⁶. Both of them are available as open-source implementations, as well as services offered by the major cloud vendors.

3.4.2 Serverless MapReduce

Due to its inherent scalability, serverless is well suited to host this type of architectural model. Amazon recently proposed a serverless architecture for MapReduce⁷ jobs using S3 for data storage and Lambda for computation. The proposed architecture offers various benefits over the conventional MapReduce libraries. It requires no operational effort to build the

⁵<https://hadoop.apache.org/>

⁶<https://spark.apache.org/>

⁷<https://aws.amazon.com/blogs/compute/ad-hoc-big-data-processing-made-simple-with-serverless-mapreduce/>

architecture, thanks to the abstraction provided by the serverless platform. It reduces the number of components, lowering the learning curve for developers. The serverless environment allows a reduction in costs as well since the pricing is per execution rather than per instance. Figure 3.3 illustrates the architectural components of the serverless MapReduce model.

Chapter 4

Environment

The system described in this paper is developed within the context of Lahti's CitiCAP project (Citizens cap and trade co-created)^{1,2}. This project has multiple goals: reduce transport-related emissions, collect and publish mobility data, and develop novel transport services for citizens. To achieve this, the EU's Urban Innovative Actions initiative has granted 4.7 million euros to fund the project. The CitiCAP Project promotes the shift to more sustainable, public transportations from private cars. The use of private transportation is enhanced in the city of Lahti by the scarcity of mass transit options. As Lahti, many other medium-size European cities suffer from this problem. The CitiCAP project stands as a testing ground for smart mobility solutions to reduce traffic CO₂ emissions.

The project intends to create a Personal Carbon Trading (PCT) scheme for mobility in an attempt to reduce traffic emissions in the Lahti area. PCT allows citizens to receive benefits, such as discounted public transport tickets or bicycle repair services, in exchange for smart mobility choices.

The aim of the system under investigation is to generate a precise estimation of the produced CO₂ relative to the mobility of a person.

Such application presents some challenges from the perspective of users

¹<https://www.smartlahti.fi/citicap/>

²<http://www.uia-initiative.eu/en/uia-cities/lahti>

involvement.

As noted by Ganti et al. [12], users may be reluctant to participate in such experiments due to the utilization of privacy-related information detected from their devices. Motion sensors and location data are used to estimate the mode of transportations. Location data, however, could also be used to deduce individuals' private information, such as home and work locations. Although privacy is user specific (privacy perception differs among individuals), appropriate measures have to be considered when presenting the data to third parties. In order to build location-based statistics, for example, anonymization techniques are required. Noise addition can be used to masquerade real paths. Locations can be reported as a region instead of a latitude-longitude pair, in such a way that a single user is unidentifiable as a person.

A second compelling problem is the significant resource utilization demanded to the device. The utilized sensors, GPS in particular, cause rapid battery depletion. The network is also required to communicate with the cloud infrastructure. Users might judge these limitations unacceptable and abandon the platform. In order to overcome such drawbacks and retain participants, projects need to devise appropriate incentives.

Recent works, such as that of Gabrielli et al. [11], describe possible solutions and research directions to increase user adoption. They identify the social component as a critical component in this sense. Users are more willing to change and improve their behavior if they can compare themselves with other people in their social network. Furthermore, they received positive feedback about challenges. Challenges encourage users to improve their behavior by giving them a goal to achieve. They note, however, that the success of persuasion goals differs among user groups, therefore challenges need to be tailored to the user.

Chapter 5

Methods

The developed system contains two main components: a mobile app (described in Section 5.1), which implements a transport mode recognition engine; the data processing part (the novel contribution of this work, introduced in Section 5.2 and thoroughly described in chapter 6), which integrates GPS information to the transport modes in order extracting reliable CO₂ emission estimation from the mobility data.

5.1 Mobile app

This Section briefly describes the mobile app and its transport mode recognition engine.

The mobile app collects data from the smartphone sensors and presents processed data to the users. It collects data from two sensors, accelerometer, and GPS. Accelerometer data is processed locally on the device with an implementation of the work by Hemminki et al. [13]. This approach offers good performance and more importantly, allows to identify the principal public transportation vehicles, which is required by the nature of our purpose. This data is processed locally due to the high quantity of data coming from the sensor, making it impractical to send over the network. The utilized implementation transforms the sensor data to a series of labeled

periods with the most likely activity recognized. Activity recognition data is then sent to the back-end along with GPS information.

5.2 Serverless data processing

Once the back-end receives the data, it is stored and processed.

The processing consists of a series of filters and refinements of the data, implemented as a sequence of rule-based transformations. The primary function of the data processing is to refine the sequence of trips and improve their categorization. Section 6.3.2 explains in detail how this task is accomplished.

Along with mobility data refinement, the back-end offers data analytics functionality. Different kinds of statistics can be computed on the data, such as the CO₂ emissions for a user and its variation in time, the average CO₂ emissions of a user group, or the number of users who have utilized public transport on a given day. Data analytics employ an approach that takes inspiration from the MapReduce model introduced in Section 3.4. This approach considerably accelerates the computation of statistics, even in the case of complex operations.

Chapter 6

Implementation

This chapter discusses how the system has been implemented. First, the overall architecture is explained in Section 6.1, it lists the components of the systems and explains how they interact with each other. After this, the following sections consider each component and describe them in detail. Section 6.2 explains how the data collection is conducted, what information is collected and the challenges that can arise from the data. Section 6.3 describes how the data is processed when a user requests it. It states the steps used to process the data and discusses their purpose. Finally, Section 6.4 explains the architecture used for statistics generation.

6.1 System overview

The system contains three main components: the data gathering, the data processing and the statistics generation.

Data gathering depends on the usage of a mobile application by the users. The app collects various data from the phone sensors and sends it to a cloud endpoint for data storage. The incoming data triggers the processing algorithm, which will be explained in detail in Section 6.3. The data can be used, for instance, to compute statistics per user or globally in order to gain insight into their behavior.

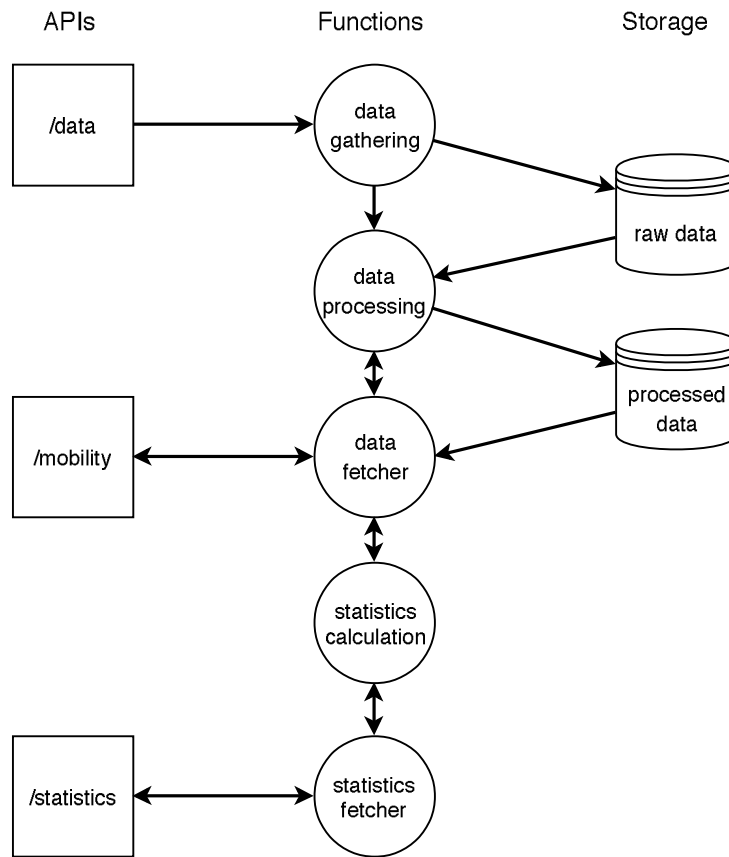


Figure 6.1: A general view on the architecture. Square elements represent the API endpoints, circle elements represent serverless functions and cylinder elements depict data storage components.

Figure 6.1 provides a high-level view of the interactions of the components. In the next sections, these will be explained more in detail.

The building blocks, which compose the architecture, are of three types:

- **API endpoints:** They are the interface that users employ to interact with the system. They trigger the functions needed to accomplish the desired operation and provide a safe environment isolated from the rest of the components.
- **Serverless functions:** They provide the logic. There are two kind of functionality they can provide: coordinators, which are triggered di-

rectly by the API endpoints, and processors, which are usually long-lasting functions, which interact with the data storage and perform data transformation. Certain functions can fall in both of these two categories (e.g., data gathering).

- **Data storage:** This component provides durable storage of data. It is logically decoupled from the functions.

6.2 Data gathering

As illustrated in Figure 6.2, when the data is sent to the API endpoint, the data gathering function is triggered (*step 1*). The goal of this function is to store the incoming data in the appropriate space (*step 2*). Before storing, the data is validated according to some predefined format to avoid parsing errors in subsequent phases of the processing. As a final step, the function triggers the data processing (*step 3*), that will perform some analysis and computations on the new data and store its output for later reuse (*step 4*).

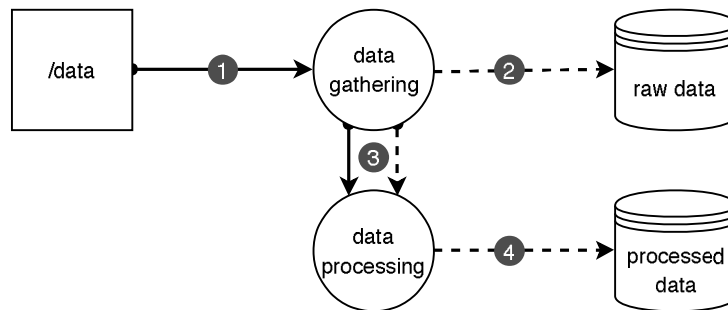


Figure 6.2: The events/data flow of the data gathering part. Solid lines represent events, while dashed lines symbolize data reads/writes.

6.2.1 Data types

The data collected from the device can be subdivided into two categories:

- **Transport mode data:** This is the core data used by the system. It consists of a list of modes of transport discovered by the smartphone

application exploiting motion sensors embedded in the device (e.g., accelerometer). The raw sensor data is processed locally on the device through a lightweight implementation of the algorithm conceived by Hemminki et al. [13]. One batch of transport mode data includes many legs. Legs are units of transport, as recognized by the algorithm. Each leg carries information about the discovered mode and the temporal beginning and end.

- **Location data:** It provides a way to enrich the transport mode data. It enables the computation of the distance traveled during each leg and the visualization of the data on a map. Location data requires careful handling since it contains sensitive user information. Location data mainly includes the time of the observation, latitude, and longitude of the device and the accuracy estimation of these values. It can also include other information provided by the positioning system of the smartphone, such as speed, altitude or position source.

The size of this data amounts to 35kB per user per day for transport mode data and about 190kB for location data. This amount of data is negligible compared to the raw accelerometer data, which can reach 300MB per user per day.

6.2.2 Data challenges

The raw data sent from the device does not always contain clean, correct and continuous data as it would be desirable. This Section will explicate the major challenges encountered during the data collection.

Phone operating system

The benchmark app has been build for the Android operating system. It is composed of the transport mode detector and an interface that provides some visualization of the data present in the system. Android phones provide a wide range of different behaviors in resource management. Vendors

can tweak their version of the operating system to prevent the application from running long task in the background in order to save battery. This limitation has proven a tough challenge for this application, which needs to keep the phone awake for long periods to register transport activities. Some effort has been put into minimizing the resource consumption of the app and provide a decent user experience.

Location data

The data read by the smartphone application is sometimes unreliable. The majority of smartphones employ some heuristics to fuse multiple sources of location information and can sometimes result in incorrect readings. The location sources commonly used include GPS, Cell-ID (i.e., position of the surrounding cellular base-stations) and Wi-Fi. While GPS provides high accuracy, at least for this application, it entails a number of disadvantages: it requires user permission since it includes sensible information, it considerably increases the battery consumption of the device and, perhaps more importantly for this application, it is unavailable in certain environments, where the satellites are occluded. Cell-ID helps in certain situations, where GPS is inaccessible but has reduced accuracy. Wi-Fi is very limited in scope, but the spread of this network and the mapping of these can help to substantially improve the location accuracy. We observed, moreover, that the presence of mobile hot-spots can break this scheme. While hot-spots are associated with a fixed location by the location service, they can be moved by the owner, thus resulting in completely erroneous observations.

6.3 Data processing pipeline

This Section describes the interaction of the cloud components and the series of transformations that is used to process the raw data retrieved from the device sensors.

6.3.1 On-demand processing

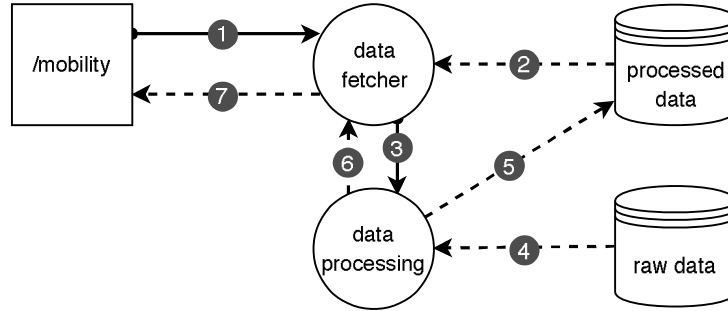


Figure 6.3: The events/data flow of the data processing part. Solid lines represent events, while dashed lines symbolize data reads/writes.

Figure 6.3 shows how the components communicate in case a user requests some mobility data. After the request has been dispatched, the data fetcher function is triggered (*step 1*). This function tries to retrieve pre-computed mobility data from the storage (*step 2*). In case the data is present in the storage, the function will directly return it to the client (*step 7*). Conversely, if the data has never been previously computed, it will trigger the data processing function (*step 3*). This function will read the raw data (*step 4*), as saved by the data gathering function, and process it as explained in Section 6.3.2. After the processing completes, the result is cached for later reuse (*step 5*) and returned to the data fetcher (*step 6*) that can finally return it to the client (*step 7*).

6.3.2 Data processing algorithm

The algorithm used to process the timeline can be decomposed into a series of semantically different steps, which will be explained in the following. Each step refines the labeled data or adds information to the dataset. The processing has been implemented as a rule-based algorithm in order to retain transparency in the process.

Raw data retrieval

The source data employed by the algorithm must include both location and transport mode data. The correct data points are retrieved according to the requested time interval. After data retrieval, the available dataset includes these two separate data types.

Location data filtering



Figure 6.4: Location sequence a) before and b) after the filtering procedure.

The location data is filtered based on an accuracy threshold and the unlikely locations are trimmed from the dataset. The accuracy threshold ensures that the least accurate points are discarded. This filtering guarantees that distance and speed estimations are less prone to noise in the data.

As discussed in Section 6.2.2, the location data can be very inaccurate when the satellite data is unavailable. Figure 6.4 shows an illustra-

tion of this problem. In 6.4(a) some false locations are present. The location sequence is analyzed to remove jumps created by the source fusion performed by the location service. Heuristics are used to accomplish this filtering. The metrics considered include estimated distance, estimated speed, estimated acceleration, change in direction, number of wrong points. These metrics can be used to identify single points distant from the real path and points close to each other, which occur after a sudden location jump. The number of points is considered to avoid the cancellation of long sequences (e.g., after a flight, where the location data is lacking and the speed is considerable). This step only analyzes location data.

Data fusion

The next step fuses together the location and the transport mode data. Each location observation is associated with a leg if the leg start and stop timestamps include the location timestamp. Furthermore, the observation directly preceding (respectively, following) is included in the leg if it is closer with respect to time than the first (last) location sample. A leg may contain zero location samples in its time interval, in this case, the previous and the following locations are included in the leg. After this step, the locations and transport modes are fused into a single dataset. The data fusion allows estimating some statistics on each leg, such as speed, distance and the number of location points in the leg.

Legs relabeling

After the data fusion step, legs are relabeled according to some heuristics. Figure 6.5 illustrates of the decision flow used to relabel the legs. The thresholds used in this step of the algorithm have been chosen empirically.

Speed is calculated as sample average of the speed estimation coming from the location data, if at least 90% of the samples include such estima-

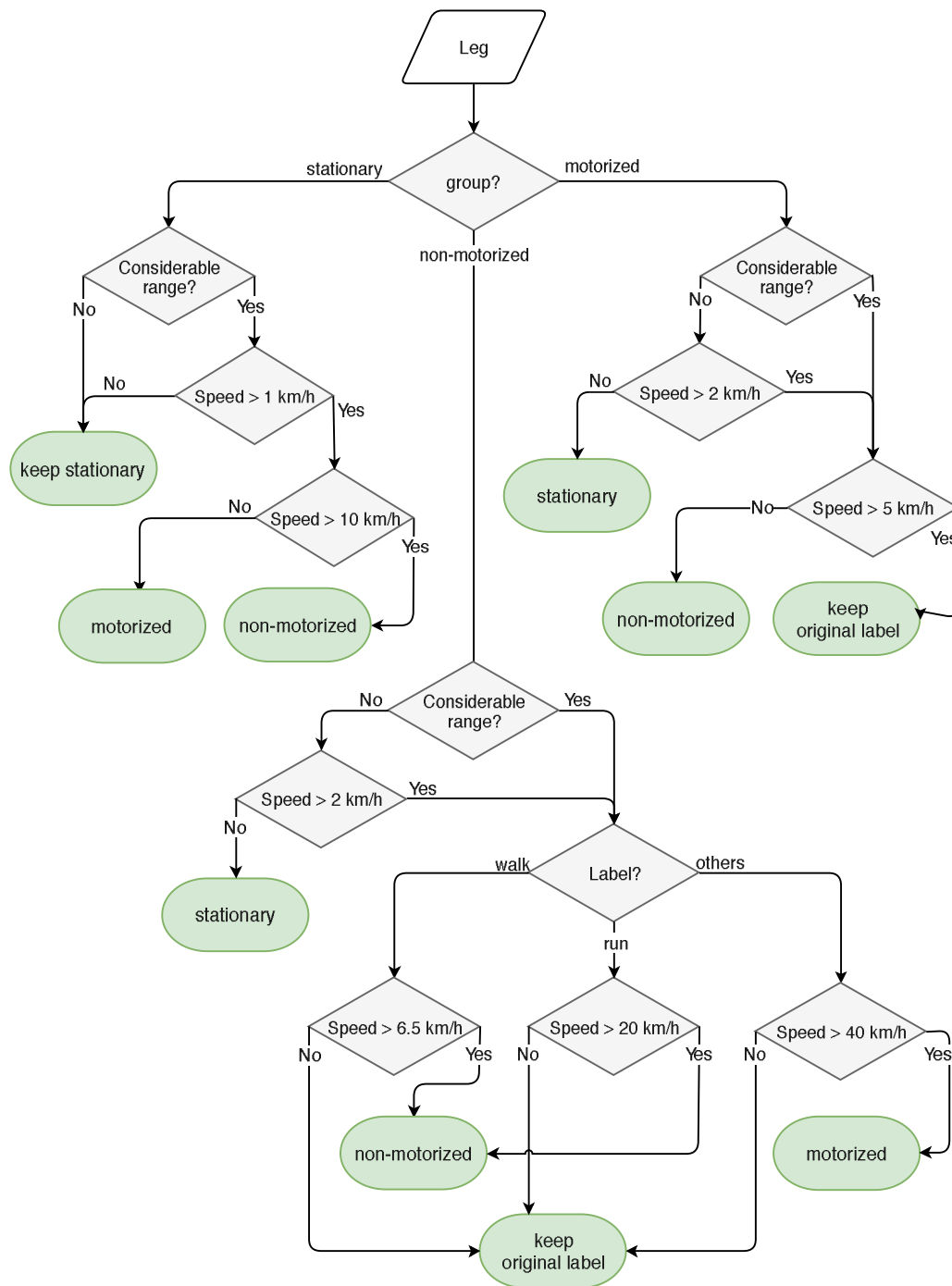


Figure 6.5: The decision graph used for relabeling.

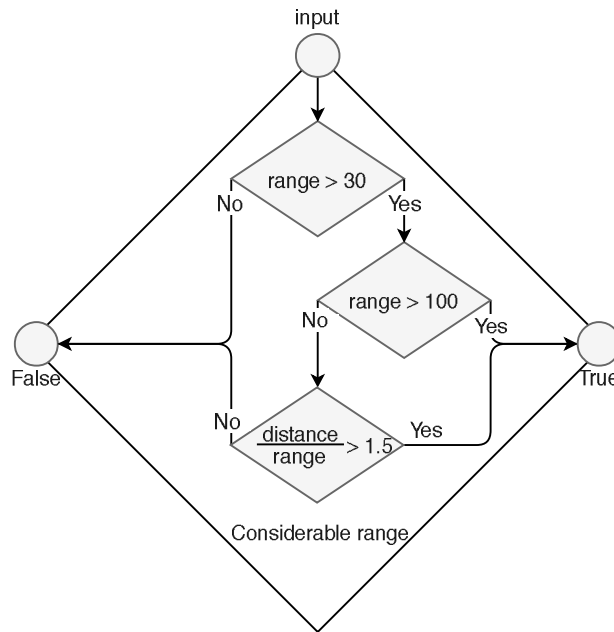


Figure 6.6: Expansion of the "Considerable range?" decision. Distance and range are computed respectively with the formulas in equations 6.3 and 6.2.

tion, otherwise the following formula is used:

$$\frac{t_{stop} - t_{start}}{\sum_{i=1}^{N-1} dist(l_i, l_{i+1})} \quad (6.1)$$

where t stands for the timestamp, N is the number of locations included in the leg, l_i is the i^{th} location sample and $dist(x)$ is the distance on the Earth surface.

Range is calculated as the distance on the Earth surface between the two furthest locations in the leg:

$$\max_{i,j \in [1,2,\dots,N]} (dist(l_i, l_j)) \quad (6.2)$$

and distance, the total covered distance, is computed as sum of piece-wise distances:

$$\sum_{i=1}^{N-1} dist(l_i, l_{i+1}) \quad (6.3)$$

with the same notation used in the speed equation.

The expression “considerable range?” is a composite expression that is used for the sake of readability in Figure 6.5. Its decomposition is explained in the graph in Figure 6.6.

This step typically improves the classification of the legs, since fused data has access to location data along with transport mode data.

Legs merging

In the merging step, each leg is evaluated together with the neighboring legs and they are pooled based on their similarity. Similarity is calculated as the cross-product between vectors including distance and duration.

$$\text{sim}(L_i, L_j) = \begin{pmatrix} L_i[\text{duration}] \\ L_i[\text{distance}] \end{pmatrix} \times \begin{pmatrix} L_j[\text{duration}] \\ L_j[\text{distance}] \end{pmatrix} \quad (6.4)$$

where L_i is the i^{th} leg,

If the similarity is deemed too low (less than 0.15) the legs are not merged.

In addition to the similarity, other two criteria are considered: time difference between the legs and duration of the resulting legs. The former is needed because the data timeline can have holes and not provide continuous coverage of time. Therefore, if there is a temporal jump between the legs, they have to be considered separately. In this case, a connecting segment is added to the timeline. The latter is a way to cluster legs together and yield a cleaner result. Legs during less than one minute are merged to hide micro-movements and considerably reduce the number of legs.

After merging the legs, a common label for the group has to be chosen. The candidate labels are those of the legs in the pool. They are ranked by duration of the legs, and the label associated with the highest duration is chosen to represent the pool. To reduce the probability of aggressive labels, a softer classification is considered. In the case that the most probable label is associated to less than 55% of the total duration, the label is judged too weak and the label is chosen from the group labels (recall that the classification scheme is hierarchical).

This step allows to considerably reduce the number of legs. The Transport Mode Detection (TMD) algorithm utilized in the mobile application is quite verbose and is able to extract transport modes with a high frequency. The merging step is necessary to clean the timeline from short segments.

Legs cleaning

The cleaning of the legs aims to simplify the geometry of the legs. The geometry is here intended as the sequence of location observations associated with the leg. Three main operations are performed in this step.

Duplicate locations are removed from the geometry. The device location service may occasionally provide duplicate locations in an attempt to fill periods in which no fresh data is available.

The geometry of stationary legs is reduced to a single point by averaging the locations in the leg.

Subsequent legs are connected to each other by adding the first location (respectively, the last location) to the end of the previous leg (respectively, the beginning of the following leg). This connection ensures that no visible holes are present when rendering the geometry. The criterion used to choose whether to connect the beginning or the end of the leg is the label of the considered legs. The timeline is predominantly a sequence of mobility legs (i.e., non-stationary trips) interleaved by stationary periods. Only the mobility legs are considered when adding this kind of connecting points. After this step, the geometry of the legs is substantially more concise. The most substantial reduction occurs on long stationary periods, such as the night period, when the phone is left still but keeps collecting location data.

Public transportations modes validation

Motorized legs are compared to public transportations routes to improve the accuracy of the classification. Candidate matches are retrieved through

the Digitransit Routing API¹. The Routing API returns multiple alternative routes providing several details for each trip, including start and end locations and time of departure. When more than one match is returned, four parameters are evaluated to choose the best match to the recognized trip: distance between the recognized and the alternative start and stop stations (p_{dist}), difference between the recognized and the alternative departure times (p_{time}), percentage difference between the recognized and the alternative trips durations (p_{dur}), and geometry similarity (p_{path}). Each of this parameters is transformed into a probability (the mapping functions have been chosen in the form of sigmoid functions), and their multiplication is used to estimate the likelihood of the GIS alternative. The probability estimate of the best alternative is compared to a threshold as in Equation 6.5; if it is considered compatible, the original leg label is corrected to the new one and the geometry is substituted with the path traveled by the public transport.

$$p_{\text{dist}} * p_{\text{time}} * p_{\text{dur}} * p_{\text{path}} > 0.75^4 \quad (6.5)$$

Geometry similarity is calculated with the Fréchet distance. This measure has already been used in research for measuring similarity of GPS trajectories [6, 21, 24]. The Fréchet distance considers both location and order of the points along the curves. To understand it the following analogy is commonly used: suppose a man is walking a dog. The first curve represents the path followed by the man and the second the path followed by the dog. The speeds of the two can be different and vary but they are not allowed to move backward. The Fréchet distance between these two curves is then the minimum length of leash necessary to connect the man and the dog.

Metro and train trips benefit from this step because the location service works intermittently along the routes, due to poor reception. The routes

¹<https://digitransit.fi/en/developers/apis/1-routing-api/routes/>

geometry is therefore improved. Bus trips, on the other hand, suffer more from mislabeling, due to the intrinsic similarity to cars. Their benefit is more related to relabeling and validation.

The Digitransit Routing API is used in this design due to its free and open source software (FOSS) nature. However, its data covers the sole Finnish territory, limiting the public transport validation to this region. In case of expansion in other countries, a different GIS system would have to be utilized. An alternative could be that of utilizing Google Transit API², which covers a long list of countries³.

Path smoothing

The final step of the legs processing consists of smoothing the geometry. This reduces the noise in the data and to make the path more appealing when rendered on a map.

First, the geometry is up-sampled by a factor of two interpolating existing data. Then, it is low-pass filtered with a sliding window of size three.

Finally, the number of locations is reduced through the Ramer-Douglas-Peucker [7, 25] (RDP) algorithm. The RDP algorithm aims to fit a simpler curve to the sequence of points by removing all points that fall within a predefined distance interval to the simplified curve. This approach ensures a reasonable smoothing effect while retaining the unique characteristics of the path and maintaining an acceptable number of points.

Output format conversion

The process concludes with the conversion of the data structures to a serialized version, ready for storage or consumption by users. All the numeric values are rounded to an appropriate precision (e.g., two decimal places for distance and duration, five decimal places for location points).

²<https://developers.google.com/transit/>

³<https://maps.google.com/landing/transit/cities/>

Transport Mode	g CO ₂ /km			
	Manufacture	Manufacture of energy or fuels	Use of fuel	Total
bicycle	5.0	0.0	0.0	5.0
train	5.0	11.7	0.0	16.7
metro	5.0	32.6	0.0	37.6
bus	6.5	3.4	34.5	44.4
tram	5.0	43.4	0.0	48.4
car	15.0	36.0	159.0	210.0

Table 6.1: The factors used to compute the produced CO₂. These values include the carbon dioxide production during the entire life-cycle of the vehicle (i.e., vehicle manufacturing and fuel production and consumption.). Data has been aggregated from VTT LIPASTO⁴ and EU directive 2009/28/EC⁵.

During this process, the carbon emissions are computed for each leg. This is accomplished by multiplying the distance traveled during the leg with the respective CO₂ production factor related to the mode of transport. These values are listed in Table 6.1.

General considerations

The previous paragraphs describe at a high level the process that the raw data undergoes. Besides to these steps, an additional filtering is used after each of them: Subsequent legs with the same labels are merged to clean the timeline from redundancy.

6.4 Data analytics

Data can be used to calculate statistics about the users. Each user can gain insight into his/her mobility data and use this awareness to improve his/her carbon footprint. Projects, as the Citicap one can use user statistics to incentivize the use of green forms of transport, such as bicycles and public transportations over private vehicles.

In order to generate this kind of data, the system has to analyze a large quantity of data. The processing architecture used for this operation takes inspiration from the MapReduce programming model. As discussed in Section 3.4, it enables the elaboration of large quantities of data by parallelizing the work into mappers and reducers.

In this case, the statistic generating function, assume the role of mappers. The statistic fetcher function assumes the role of driver and reducer. The output of each of the statistics generating functions is fed into the operation driver, which merges all the single outputs and return an aggregate

⁴<http://lipasto.vtt.fi/en/index.htm>

⁵<https://eur-lex.europa.eu/legal-content/en/ALL/?uri=CELEX:32009L0028>

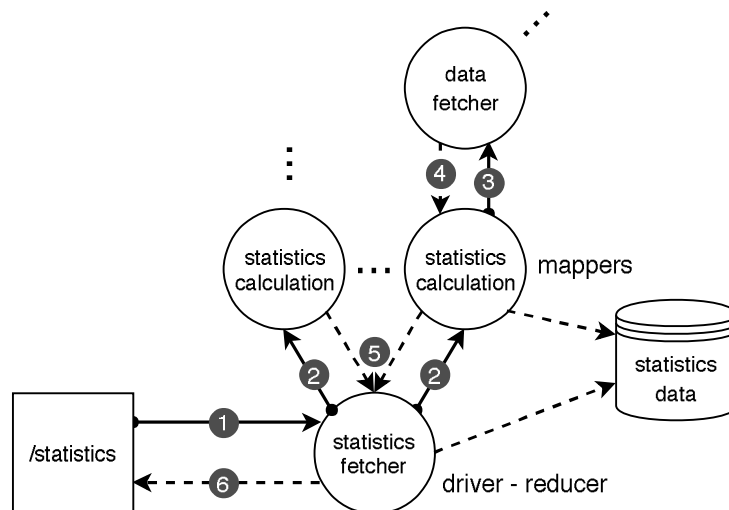


Figure 6.7: The architecture used to perform the task parallelization of functions.

summary of the statistics.

The major difference with respect to the general MapReduce scheme is that the request is synchronous and is consumed directly by the requester, instead of being only stored. This is required since the functions are linked to an API endpoint that returns data directly to the invoker. This synchronous scheme implies that the function that the caller invokes is also the one that returns the data. In place of a straight data flow, this involves a backward scheme of function calls in order to achieve the desired result. In a serverless environment, where each function is billed by the execution time, this implies an increase in costs. This increase can be limited, however, by reducing the performance requirements of the driver, that spend most of the time idle.

Figure 6.7 shows how the functions are communicating. When a user requests statistics, the driver is run (*step 1*). It takes care of requesting the needed data from the mappers (*step 2*), which recursively call the data fetcher for data retrieval (*step 3*). They return the processed data as explained in Section 6.3 (*step 4*). The result is fed into the mappers, in the Figure named “statistics calculation” functions (*step 5*). As soon as the mappers terminate their computations and return their outputs, the driver combines their output into a single dictionary and sends the final computations to the user (*step 6*).

Chapter 7

Evaluation

This chapter evaluates the service developed. Section 7.1 estimates the operational costs of the platform, focusing specifically on the serverless functions. Section 7.2 reports the results of the test conducted and compares the results with the TrafficSense project.

7.1 Cost analysis

Operational cost for the service can be dissected into a set of costs for the services used. This analysis is centered around the cost of executing the functions. The data gathering function and the data processing function are considered because they are available to the general users. The data analysis function is currently limited to internal use and therefore omitted.

Costs for data storage are omitted since they would be the same in a comparable service built in a traditional architecture. Furthermore, costs related to the platform development, such as salaries, are omitted as well, even though they are considerable for a large platform. Serverless abstraction of infrastructure allows to considerably reduce this cost in a real scenario.

Figure 7.1 depicts the pricing models of the principal cloud providers. The pricing strategies are rather similar: providers charge based on the

Price comparison of major serverless providers [\\$]

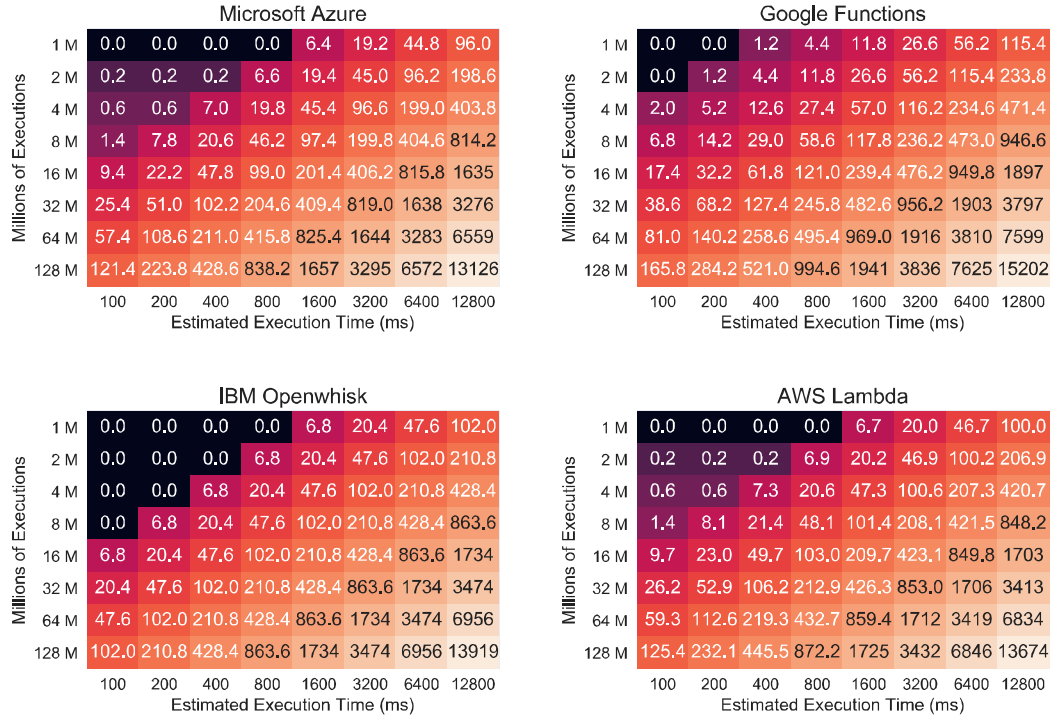


Figure 7.1: Cost comparison among the main cloud vendors. The prices shown are for a fixed memory of 512 MB.

number of executions, the duration of each execution and resources allocated for an execution. Price variation among providers is practically negligible.

The pricing model of the cloud vendor is similar. They provide a conversion from execution time given a resource setting. Typically, users may only modify the memory metric. The memory setting also influences other infrastructure specifications, such as the CPU; with the exception of Google Cloud Functions where users are able to choose the CPU setting explicitly. In all the other cases, as memory increases, CPU scales accordingly.

Figure 7.2 illustrates how cost and execution time varies as a function of the memory settings. For each of the two functions, a representative request has been chosen to model the execution time. The cost has been

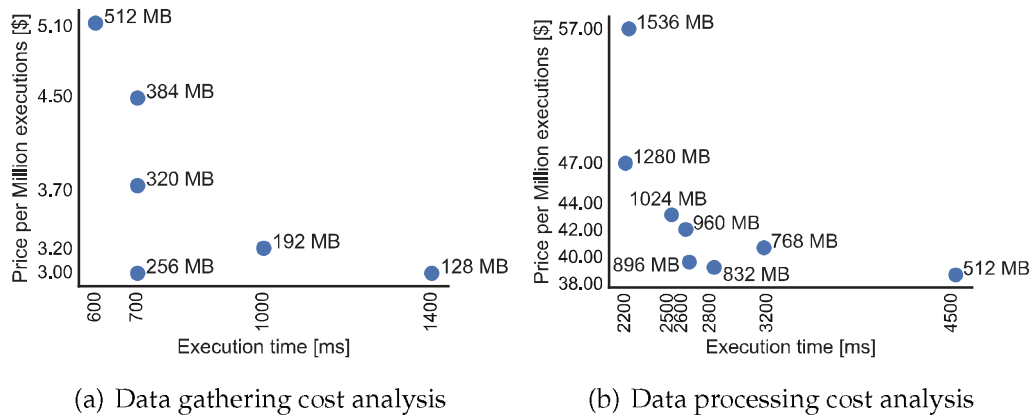


Figure 7.2: Cost analysis for the two main functions of the platform: a) data gathering and b) data processing. The better results minimize both the price and the execution time, therefore are those closer to the bottom-left part of the plots.

calculated with the pricing conversion of the AWS platform. The optimal memory setting for the data gathering function has been chosen as 256 MB, while for the data processing as 896 MB. The two values represent the best compromise in terms of execution speed and projected cost.

Function	Estimated price per Million executions	Average execution Time	Price per user per month
Data Gathering	2.99 \$	700 ms	0.0024 \$
Data Processing	39.57 \$	2650 ms	0.0120 \$

Table 7.1: Optimized values for execution time and price for the functions.

Table 7.1 reports the estimated price per million executions and the relative average execution time. The estimated price per user per month has been estimated in 0.014\$. The price scales linearly for each new user. This value has been calculated considering the estimated traffic of users having deployed the platform on the public cloud. The data gathering function is executed approximately once per hour. This invocation pattern follows from the client application settings; it is set to upload new data with an hourly pace. The data processing function has a more complex execution

pattern. The first tests show that its invocations occur with irregular intervals, predominantly during the daytime. The average number of daily invocations amounts to ten per user. The invocations are largely grouped in multiple requests. In order to minimize the computations executed by the system, it offers a simple caching mechanism. Multiple requests for the same data are redirected to the cache. This simple tool considerably reduces the computations and the response time.

7.2 Accuracy

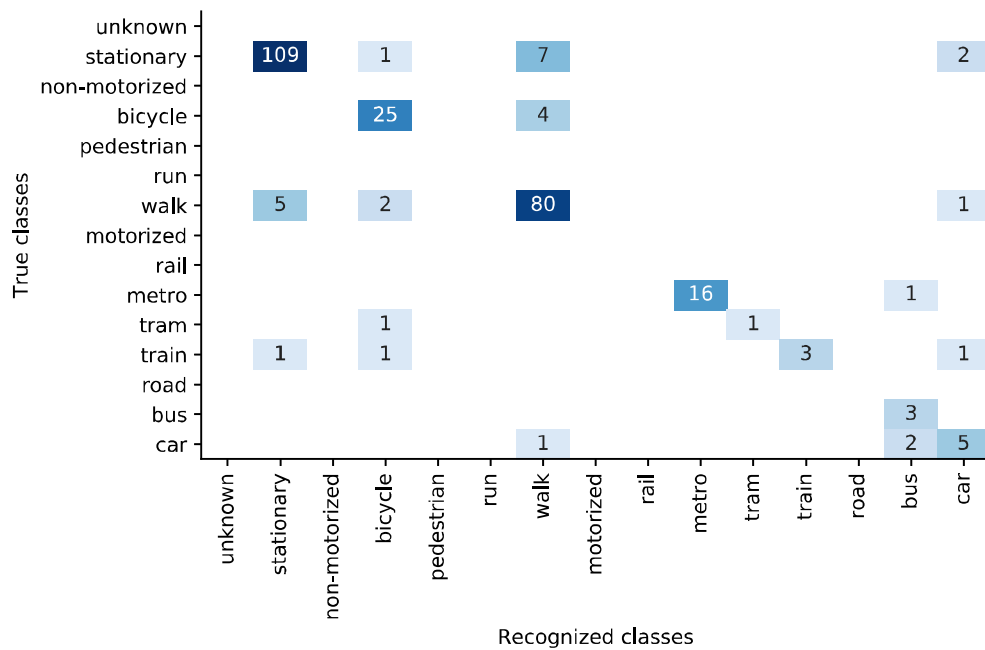
The performance of the system has been measured on some significant days of users. A total of 163 hours of data is evaluated based on the correctness of the classification. The data has been sampled considering a reasonable coverage of the transport modalities. Figure 7.3 illustrates the confusion matrices between the classes.

Figure 7.3(a) is the confusion matrix of the recognized modalities using exclusively the activity recognition engine on the smartphone, which employs only the accelerometer. The actual classes have been identified with annotations by the users. Due to the high responsiveness of the algorithm, many stationary periods include small motions, which are recognized as such. However, since they only account for small movements, they are irrelevant to the mobility and therefore discarded.

Figure 7.3(b) shows the confusion matrix for the processed modalities in the cloud. The number of trips is significantly reduced, retaining only the relevant trips (i.e., considerable duration and distance covered). After the processing, the confusion is also considerably reduced. Precision has increased from 71.0% to 89.9%, while recall has improved from 36.2% to 89.0% and f-score has risen from 47.3% to 89.0%. The Figure shows that the majority of the confusion arises among the motorized classes and between ‘stationary’ and ‘walk’. The latter derives mainly from short movements before or after the actual trip. These movements have not been marked in



(a) Off-line activity recognition results



(b) Cloud activity recognition results

Figure 7.3: Confusion Matrices. a) Confusion matrix for the activity recognition run on the smartphone (accelerometer only). b) Confusion matrix after the data processing done in the cloud (enriched with GPS and public transport information.)

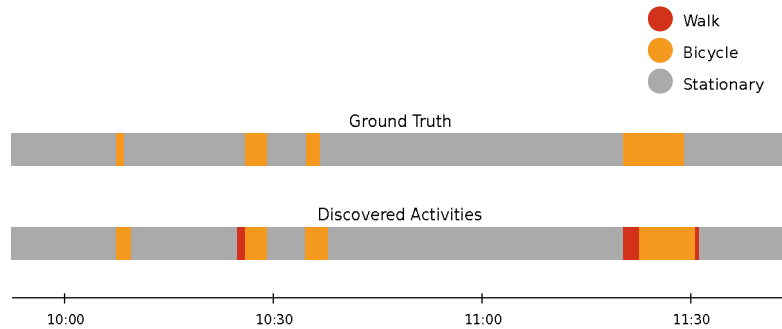


Figure 7.4: Timeline excerpt comparison. Short movements have been identified by the algorithm although they are absent in the ground truth.

the ground truth, but have been recognized by the algorithm. Figure 7.4 shows an example of a sequence containing such mistakes.

Table 7.2 show the results of the recognition limited to motorized trips. Out of 29 motorized trips, 23 have been correctly labeled (79.3%); four are partially correct (5, 11, 13, 16), but split into different modalities, two are incorrect (17, 19). Sample 13 has been divided into a sequence of ‘car’, ‘walk’, ‘car’. The middle segment is probably due to a slow traffic period, which has been relabeled to ‘walk’ when comparing to speed thresholds. Sample 5, 11 and 15 are split into multiple labels (‘train’, ‘stationary’, ‘bicycle’, ‘car’ for Sample 5, ‘bicycle’, ‘tram’ for Sample 11, and ‘metro’, ‘bus’ for Sample 15), although belonging to a single ride. The change of modality probably originated from user interaction with the phone, walking inside the vehicle or lack of GPS points. Samples 17 and 19 have been labeled as ‘bus’, despite belonging to the ‘car’ class. These two classes have proven difficult to distinguish by the underlying activity recognition algorithm, so as ‘train’, ‘tram’ and ‘metro’ [13]. They have not, however, been validated by the public transport API.

18 out of 23 (78.3%) public transport rides have been recognized correctly by the public transport API, yielding a correct line code. Samples 5, 11 and 15 has been partially recognized, as described above. Sample 6 has not been recognized, while Sample 16 has been mislabeled; the errors are due to anomalies in the public transport timetable.

Sample	Correct		Discovered		Departure		Arrival		Distance covered [km]
	Modality	Line	Modality	Line	Time	Location	Time	Location	
1	metro	M	metro	M2	18:06:46	Otaniemi	18:15:40	Ruoholahti	7.4
2	metro	M	metro	M1	18:18:25	Ruoholahti	18:24:59	Kluuvi	2.18
3	metro	M	metro	M2	22:29:00	Kluuvi	22:43:45	Otaniemi	9.53
4	metro	M	metro	M1	19:13:38	Otaniemi	19:25:57	Kluuvi	9.06
5	train	I	train	I	19:32:37	Kluuvi	19:52:42	Koivukylä	19.56
5	train	I	stationary		19:52:42	Koivukylä	20:10:07	Koivukylä	0.0
5	train	I	bicycle		20:10:07	Koivukylä	20:12:36	Koivukylä	0.43
5	train	I	car		20:12:36	Koivukylä	20:14:52	Lentokenttä	3.63
6	train	P	train		21:07:00	Lentokenttä	21:24:37	Kluuvi	16.17
7	metro	M	metro	M2	21:50:36	Kluuvi	22:02:30	Otaniemi	8.97
8	metro	M	metro	M2	19:13:17	Otaniemi	19:25:33	Kluuvi	9.66
9	metro	M	metro	M1	23:24:46	Kluuvi	23:37:50	Otaniemi	9.04
10	metro	M	metro	M1	17:39:49	Otaniemi	17:51:25	Kluuvi	9.12
11	tram	I	bicycle		21:25:43	Taka-Töölö	21:26:51	Taka-Töölö	0.38
11	tram	I	tram	I	21:26:51	Taka-Töölö	21:35:01	Etu-Töölö	1.67
12	metro	M	metro	M2	21:42:31	Kamppi	21:52:34	Otaniemi	8.63
13	car		car		08:55:19	Vartioharju	08:57:16	Vartioharju	0.66
13	car		walk		08:57:16	Vartioharju	09:00:10	Vartioharju	0.13
13	car		car		09:00:10	Vartioharju	09:04:11	Puotila	1.04
14	metro	M	metro	M1	09:08:40	Puotila	09:37:57	Otaniemi	20.63
15	metro	M	metro	M2	16:16:46	Otaniemi	16:43:33	Itäkeskus	19.43
15	metro	M	bus	841	16:43:33	Itäkeskus	16:47:16	Puotila	1.42
16	car		car		16:49:50	Puotila	16:54:19	Vartioharju	1.45
17	car		bus		17:44:02	Vartioharju	17:53:36	Keski-Vuosaari	3.85
18	car		car		19:55:11	Keski-Vuosaari	20:00:55	Keski-Vuosaari	0.82
19	car		bus		20:17:17	Keski-Vuosaari	20:27:40	Vartioharju	4.05
20	car		car		08:58:04	Kulosaari	09:02:50	Kulosaari	1.44
21	metro	M	metro	M2	09:05:57	Kulosaari	09:17:07	Ruoholahti	7.16
22	metro	M	metro	M2	10:28:04	Ruoholahti	10:37:18	Otaniemi	7.45
23	metro	M	metro	M2	18:07:03	Otaniemi	18:28:19	Kulosaari	14.48
24	metro	M	metro	M2	17:47:55	Otaniemi	17:57:08	Ruoholahti	7.41
25	bus	8X	bus	8X	18:04:28	Ruoholahti	18:15:12	Taka-Töölö	2.41
26	bus	18	bus	14	20:19:59	Taka-Töölö	20:25:38	Meilahti	1.6
27	bus	551	bus	551	20:41:20	Meilahti	20:51:47	Otaniemi	4.96
28	metro	M	metro	M2	11:51:18	Otaniemi	12:00:25	Kluuvi	9.19
29	train	I	train	I	12:18:32	Kluuvi	12:43:57	Lentokenttä	24.05

Table 7.2: Motorized trips. Comparison of recognized activities and ground truth for motorized modalities.

Chapter 8

Discussion

The hybrid approach implemented represents a novel approach to transport mode recognition. The following sections discuss challenges and possible further improvements to the transport mode recognition (section 8.1) and the cloud implementation (section 8.2).

8.1 Transport mode recognition

The system described enhanced the transport mode discovery capability of the accelerometer-only approach based on the work from Hemminki et al. [13].

In order to improve the final accuracy, further improvements to the activity recognition engine are planned. The data processing will undoubtedly benefit from improved raw data.

In addition, further studies on decreasing the power consumption are necessary. Battery depletion stands as a compelling problem of activity recognition frameworks and represents a critical deterrent from broad adoption, and therefore has to be addressed. The leading cause of this problem is the necessity of continuously reading data from the sensors. This continuous reading prevents the device to enter into a power saving mode. Possible improvements could be obtained by introducing an

adaptive sampling rate of the sensors; stopping the data collection during stationary periods, reducing the sampling rate of GPS on slow movements and increasing it on high-speed trips.

8.2 Cloud implementation

Serverless is a novel approach for event-driven computation. It provides numerous advantages over traditional architectures, such as built-in scalability and enhanced simplicity of utilization. It, however, poses several challenges for the developers:

- The integration of other services into the serverless functions require a deeper understanding of the cloud ecosystem offered by the vendors. Few services scale appropriately as serverless functions do, therefore developers have to carefully select suitable services for all the required functionalities.
- The system work-flow and the service and function composition is not straightforward. Due to the infancy of the serverless ecosystem, limited tools are available to ease the development.
- The testing and debugging process has proven laborious, mainly due to the lack of proper tools. The abstraction of infrastructure provided by serverless complicates the debugging phase. The reproducibility of functions behaviors has proven difficult to obtain in a different environment. Although most programming languages strive to abstract from the underlying infrastructure, in a cloud environment functions can show different behaviors.

The proposed architecture can be further improved to address scalability issues. Future work includes:

- Experimenting with different approaches for the data gathering. Its behavior is barely suited to the serverless platform. It is I/O-bound,

as opposed to the preferred CPU-bound and its invocation pattern appears to be quite regular. Therefore, it could transform into a scalability bottleneck with larger user pools.

- The data processing could be executed on the device itself. This could bring some benefits such as reduced infrastructure costs for the cloud system and more privacy for users data. A possible drawback is the increased battery consumption, due to the computations. Future work could explore this trade-off.
- The stored data format can be improved and its storage footprint minimized. This aspect has not been studied in depth during the development of the platform, however, it stands as a critical point for scalability.
- The public transport GIS data is currently limited to the Finnish territory. In order for the system to express its full potential outside this boundary, similar data sources, covering other territories, have to be integrated into the system.

Chapter 9

Conclusions

This thesis describes a novel system for transport mode recognition. It consists of a hybrid architecture both in terms of data source and in data processing platform. The approach described exploits both smartphones and the novel serverless technology to accomplish mobility trips discovery. Data sources employed include accelerometer, GPS and public transports GIS (i.e., public transport timetables).

The data processing algorithm developed improves considerably the accuracy of the off-line activity recognition engine run on the smartphone: the precision improved from 71.0% to 89.9%. The main contribution of the thesis is the data fusion of accelerometer-based activity recognition data to GPS and public transports GIS data. The two data sources introduced allow to validate the discovered modalities, find incongruities in the data and add rich information to it, such as distance covered, average speed, CO₂ produced, and a path geometry.

The serverless platform has proven worthy of attention due to its elasticity and its simplicity of utilization, although with some caveats discussed in section 8.2. Despite being in its infancy, it has already seen a significant amount of attention in particular from small-sized companies and start-ups, which require agility and low development costs. Processing of big-data is well suited to the serverless platform employing techniques

such as MapReduce, which can exploit excellent parallelization capability.

The developed system is set to be a noteworthy component of Lahti's CitiCAP project, promoting cleaner transit in spite of private transportations. We hope that this project will not remain as a vain effort and that the town, as well as other municipalities, embrace its philosophy to decrease transport-related emissions.

Bibliography

- [1] 2016. *International Energy Outlook*. U.S. Energy Information Administration.
- [2] 2017. *CO2 emissions from fuel combustion*. International Energy Agency.
- [3] Andrew Baird, George Huang, Chris Munns, and Orr Weinstein. 2017. Serverless Reference Architectures with AWS Lambda; Overview and Best Practices. (2017).
- [4] Ioana Baldini, Paul Castro, Kerry Chang, Perry Cheng, Stephen Fink, Vatche Ishakian, Nick Mitchell, Vinod Muthusamy, Rodric Rabbah, Aleksander Slominski, and Philippe Suter. 2017. *Serverless Computing: Current Trends and Open Problems*. Springer Singapore.
- [5] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters. *Commun. ACM* 51, 1 (2008), 107–113.
- [6] Thomas Devogele, Laurent Etienne, Maxence Esnault, and Florian Lardy. 2017. Optimized Discrete FrÉchet Distance Between Trajectories. In *Proceedings of the 6th ACM SIGSPATIAL Workshop on Analytics for Big Geospatial Data (BigSpatial'17)*. ACM, 11–19.
- [7] David H. Douglas and Thomas K. Peucker. 1973. Algorithms for the reduction of the number of points required to represent a digitized

- line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization* 10, 2 (1973), 112–122.
- [8] Geert Draijer, Nelly Kalfs, and Jan Perdok. 2000. Global positioning system as data collection method for travel research. *Transportation Research Record: Journal of the Transportation Research Board* 1719 (2000), 147–153.
- [9] Adam Eivy. 2017. Be Wary of the Economics of “Serverless” Cloud Computing. *IEEE Cloud Computing* 4, 2 (March 2017), 6–12.
- [10] Sheila Ferrer and Tomás Ruiz. 2014. Travel behavior characterization using raw accelerometer data collected from smartphones. *Procedia-Social and Behavioral Sciences* 160 (2014), 140–149.
- [11] Silvia Gabrielli, Paula Forbes, Antti Jylhä, Simon Wells, Miika Sirén, Samuli Hemminki, Petteri Nurmi, Rosa Maimone, Judith Masthoff, and Giulio Jacucci. 2014. Design challenges in motivating change for sustainable urban mobility. *Computers in Human Behavior* 41 (2014), 416–423.
- [12] Raghu K. Ganti, Fan Ye, and Hui Lei. 2011. Mobile crowdsensing: current state and future challenges. *IEEE Communications Magazine* 49, 11 (2011).
- [13] Samuli Hemminki, Petteri Nurmi, and Sasu Tarkoma. 2013. Accelerometer-based Transportation Mode Detection on Smartphones. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems (SenSys '13)*. ACM, Article 13, 14 pages.
- [14] Samuli Hemminki, Petteri Nurmi, and Sasu Tarkoma. 2014. Gravity and Linear Acceleration Estimation on Mobile Devices. In *Proceedings of the 11th International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MOBIQUITOUS '14)*. 50–59.

- [15] Scott Hendrickson, Stephen Sturdevant, Tyler Harter, Venkateshwaran Venkataramani, Andrea C Arpaci-Dusseau, and Remzi H Arpaci-Dusseau. 2016. Serverless computation with openlambda. *Elastic* 60 (2016).
- [16] Mohammadreza H. Heydary, Pritesh Pimpale, and Anand Panangadan. 2018. Automatic Identification of Use of Public Transportation from Mobile Sensor Data. In *2018 IEEE Green Technologies Conference (GreenTech)*. 189–196.
- [17] Arash Jahangiri and Hesham A Rakha. 2015. Applying Machine Learning Techniques to Transportation Mode Recognition Using Mobile Phone Sensor Data. *IEEE Trans. Intelligent Transportation Systems* 16, 5 (2015), 2406–2417.
- [18] Jerald Jariyasunant, Andre Carrel, Venkatesan Ekambaram, DJ Gaker, Thejovardhana Kote, Raja Sengupta, and Joan L Walker. 2011. The Quantified Traveler: Using personal travel data to promote sustainable transport behavior. (2011).
- [19] Maria Kugler, Sebastian Osswald, Christopher Frank, and Markus Lienkamp. 2014. Mobility tracking system for CO2 footprint determination. In *Proceedings of the 6th International Conference on Automotive User Interfaces and Interactive Vehicular Applications*. ACM.
- [20] Oana Lorintiu and Andrea Vassilev. 2016. Transportation mode recognition based on smartphone embedded sensors for carbon footprint estimation. In *Intelligent Transportation Systems (ITSC), 2016 IEEE 19th International Conference on*. IEEE, 1976–1981.
- [21] Nehal Magdy, Mahmoud A. Sakr, Tamer Mostafa, and Khaled El-Bahnasy. 2015. Review on trajectory similarity measures. In *2015 IEEE Seventh International Conference on Intelligent Computing and Information Systems (ICICIS)*. 613–619.

- [22] Vincenzo Manzoni, Diego Maniloff, Kristian Kloeckl, and Carlo Ratti. 2010. Transportation mode identification and real-time CO2 emission estimation using smartphones. *SENSEable City Lab, Massachusetts Institute of Technology*, nd (2010).
- [23] Garrett McGrath and Paul R. Brenner. 2017. Serverless computing: Design, implementation, and performance. In *Distributed Computing Systems Workshops (ICDCSW), 2017 IEEE 37th International Conference on*. IEEE, 405–410.
- [24] Jinkwan Park, Taeyong Kim, Bokuk Park, and Hwan-Gue Cho. 2016. Fast Heuristic Algorithm for Similarity of Trajectories Using Discrete Fréchet Distance Measure. *KIISE Transactions on Computing Practices* 22, 4 (2016), 189–194.
- [25] Urs Ramer. 1972. An iterative procedure for the polygonal approximation of plane curves. *Computer graphics and image processing* 1, 3 (1972), 244–256.
- [26] Jesse Read, Indrė Žliobaitė, and Jaakko Hollmén. 2016. Labeling sensing data for mobility modeling. *Information Systems* 57 (2016), 207–222.
- [27] Mikko Rinne, Mehrdad Bagheri, Tuukka Tolvanen, and Jaakko Hollmén. 2017. Automatic Recognition of Public Transport Trips from Mobile Device Sensor Data and Transport Infrastructure Information. In *Personal Analytics and Privacy. An Individual and Collective Perspective*, Riccardo Guidotti, Anna Monreale, Dino Pedreschi, and Serge Abiteboul (Eds.). Springer International Publishing, Cham, 76–97.
- [28] Peter Sbarski and S Kroonenburg. 2017. *Serverless Architectures on AWS: With examples using AWS Lambda*. Manning Publications Company.

- [29] Rahul C. Shah, Chieh-yih Wan, Hong Lu, and Lama Nachman. 2014. Classifying the mode of transportation on mobile phones using GIS information. In *Proceedings of the 2014 ACM international joint conference on pervasive and ubiquitous computing*. ACM, 225–229.
- [30] Muhammad Shoaib, Stephan Bosch, Ozlem Durmaz Incel, Hans Scholten, and Paul JM Havinga. 2015. A survey of online activity recognition using mobile phones. *Sensors* 15, 1 (2015), 2059–2085.
- [31] Liang Wang, Mengyuan Li, Yinqian Zhang, Thomas Ristenpart, and Michael Swift. 2018. Peeking Behind the Curtains of Serverless Platforms. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. USENIX Association, 133–146.
- [32] Jean Louise Wolf. 2000. *Using GPS data loggers to replace travel diaries in the collection of travel data*. Ph.D. Dissertation. Citeseer.