UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

RAUL SERGIO BARTH

LDAVI
LambDa Architecture driVen Implementation

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Renata Galante

Porto Alegre
2019

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Barth, Raul Sergio
Normas para Apresentação de Dissertações do Instituto de Informática e do PPGC [manuscrito] / João da Silva. – 2019.
15 f.:il.
Orientador: Renata Galante; Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação. Porto Alegre, BR – RS, 2019.
1.ABNT. 2.Processadores de Texto 3.Formatação eletrônica de documentos. I. Galante, Renata. II. Título.

AGRADECIMENTOS

Primeiramente, eu gostaria de agradecer a minha esposa, Mira, por ser essa pessoa maravilhosa que entrou em minha vida e que sempre me ajudou a crescer e a melhorar. Agradeço a ela por sempre ter estado ao meu lado, me apoiado e me incentivado durante todo o mestrado e toda a minha vida – e por recentemente ter me dado o presente mais maravilhoso que eu poderia ganhar: meu filho Davi. Aproveito e agradeço a ele, por ter me tornado pai, e o homem mais completo que eu poderia ser.

Agradeço também a minha mãe**,** mãe, por ter me escolhido, há 27 anos, como filho, e por ter, deste então, sempre me dado todo o amor, educação, e ferramentas necessárias para que essa dissertação fosse possível. Obrigado por ter sido, durante 27 anos, um espelho para mim, e por recentemente, ter se tornado essa avó maravilhosa.

Obrigado a professora Renata, pelo qual tenho especial carinho, por ter me guiado durante todo o mestrado, e ter sido sempre compreensível com os percalços enfrentados.

LDAVI: LambDa Architecture driVen Implementation applied to Smart Mobility

ABSTRACT

Data has been playing an important role in many areas of society. It has massively increased among time and can be a powerful source of knowledge. The way data is handled, and this knowledge is extracted had also to be adapted to support this huge amount of information coming from different sources. Lambda Architecture comes to supply this need of having a Big Data architecture capable of processing both historical data and stream data. We present LDAVI, a Lambda Architecture Driven Implementation based on Lambda Architecture approach (KIRAN, 2015), a data-processing architecture for handling massive amount of data by decomposing the problem into three layers: batch layer – for historical data processing - serving layer and speed layer – for streaming processing. Main technologies used for building this architecture are Apache Hadoop, Apache Spark, Apache Impala and Apache Kafka. The main focus is to this describe this architecture as well as its implementation, as it can apply to any type of problem where one needs to store and process huge amount of data – either in streaming or batch modes. Our objective in this work is to demonstrate the powerful, capacity and feasibility of this architecture and that it can be used to approach different type of Big Data scenarios. In this work we address Smart Mobility are as our case of study to evaluate LDAVI. We analyze passengers smart card and buses GPS and stops location from the city of Schenzhen, aiming to extract passengers density and flow. Lambda Architecture is a new architectural concept that emerged with the raise of Big Data Analytics. In this work we approach and provide an implementation of this architecture, building it with the main Big Data technology stack. Although it has started being used in some areas such as search engines and platforms requiring real-time processing – such as video stream players – we demonstrate that this architecture can also bring benefits for Smart Mobility, more precisely in public transportation. Differently from related works, we approach three different types of trip: simple trip, connection trip and round trip, what makes the analysis complete and more accurate.

Keywords: Lambda Architecture, Big Data, Smart Mobility, Passenger Density.

RESUMO

Os dados têm desempenhado um papel importante em muitas áreas da sociedade. Eles aumentaram massivamente com o tempo e podem ser uma poderosa fonte de conhecimento. A forma como os dados são tratados, e esse conhecimento é extraído, também deve ser adaptada para suportar essa enorme quantidade de informações vindas de diferentes fontes. A Lambda Architecture vem suprir essa necessidade de ter uma arquitetura Big Data capaz de processar dados históricos e dados em tempo real. Apresentamos o LDAVI, uma implementação da Lambda Architecture baseada na arquitetura Lambda (KIRAN, 2015), uma arquitetura de processamento de dados para manipular uma quantidade massiva de dados decompondo o problema em três camadas: camada de lote - para processamento de dados históricos - camada de veiculação e camada de velocidade - para processamento de streaming. As principais tecnologias usadas para construir essa arquitetura são o Apache Hadoop, o Apache Spark, o Apache Impala e o Apache Kafka. O foco principal é descrever essa arquitetura, bem como sua implementação, pois ela pode ser aplicada a qualquer tipo de problema em que seja necessário armazenar e processar uma grande quantidade de dados - nos modos de fluxo contínuo ou lote. Nosso objetivo neste trabalho é demonstrar o poder, a capacidade e a viabilidade dessa arquitetura e que ela pode ser usada para abordar diferentes tipos de cenários de Big Data. Neste trabalho, abordamos a Mobilidade Inteligente como nosso caso de estudo para avaliar o LDAVI. Analisamos os cartoes de passageiros, GPS de ônibus e paradas de ônibus da cidade de Schenzhen, com o objetivo de extrair a densidade e o fluxo de passageiros. Lambda Architecture é um novo conceito arquitetônico que surgiu com o aumento da area de Big Data Analytics. Neste trabalho, abordamos e fornecemos uma implementação dessa arquitetura, construindo-a com a principal pilha de tecnologia de Big Data. Embora tenha começado a ser usado em algumas áreas, como mecanismos de busca e plataformas que exigem processamento em tempo real - como reprodutores de fluxo de vídeo - demonstramos que essa arquitetura também pode trazer benefícios para a Mobilidade Inteligente, mais precisamente no transporte público. Diferentemente dos trabalhos relacionados, abordamos três tipos diferentes de viagem: viagem simples, viagem de conexão e ida e volta, o que torna a análise completa e mais precisa.

Palavras-chave: Lambda Architecture, Big Data, Mobilidate Inteligente, densidade de passageiros

FIGURES LIST

TABLES LIST

ABREVIATIONS LIST

HDFS	Hadoop File System
IPEA	Instituto de Pesquisa Economica e Aplicada
SPTrans	Sao Paulo Transportes S/A
LDAVI	LAMBDA ARCHITECTURE DRIVEN IMPLEMENTATION

SUMMARY

1 INTRODUCTION

Data has been playing an important role in many areas of society. It has massively increased among time and can be a powerful source of knowledge. The way data is handled, and this knowledge is extracted had also to be adapted to support this huge amount of information coming from different sources. Lambda Architecture comes to supply this need of having a Big Data architecture capable of processing both historical data and stream data. We present LDAVI, a Lambda Architecture Driven Implementation based on Lambda Architecture approach (KIRAN, 2015), a data-processing architecture for handling massive amount of data by decomposing the problem into three layers: batch layer – for historical data processing - serving layer and speed layer – for streaming processing.

Our objective in this work is to implement a low-cost framework based on Lambda Architecture for bus service management, aiming to extract passengers density and flow through Smart Card, bus stop geolocation and buses GPS data. We present LDAVI, a Big Data framework based on Lambda Architecture approach (KIRAN, 2015). The main focus is to this describe this architecture as well as its implementation, as it can apply to any type of problem where one needs to store and process huge amount of data – either in streaming or batch modes. Here we apply this architecture in Smart Mobility are to demonstrate its capacity and feasibility. LDAVI is capable of analyzing passenger flow and density of public bus service, using data obtained from buses GPS, passengers' smart card and bus stops geolocation. Moreover, bus stops and city zones are categorized according to the passengers' trips purposes: *work, residential, nightlife, personal*. Therefore, this knowledge about density and flow of passengers, besides characterizing city areas and bus stops will demonstrate a real scenario of public bus demand, enabling better governance and reorganization of the service and providing a decision-making source. This work implements Lambda Architecture (KIRAN, 2015) as the processing unit for computing passenger and bus data. All computation is performed on the top of a Lambda Architecture, a data-processing architecture for handling massive amount of data by decomposing the problem into three layers: batch layer – for historical data processing - serving layer and speed layer – for streaming processing (GRIBAUDO , 2018). Main technologies used for building this architecture are Apache Hadoop, Apache Spark, Apache Impala and Apache Kafka. A case of study analyzing passenger density and flow for the city of Schenzhen, China, has been performed. In addition, this city and its bus stops have been categorized according to travel purposes.

Lambda Architecture is a new architectural concept that emerged with the raise of Big Data Analytics. In this work we approach and provide an implementation of this architecture, building it with the main Big Data technology stack. Although it has started being used in some areas such as search engines and platforms requiring real-time processing – such as video stream players – we demonstrate that this architecture can also bring benefits for Smart Mobility, more precisely in public transportation. Differently from related works, we approach three different types of trip: simple trip, connection trip and round trip, what makes the analysis complete and more accurate.

The scope of our case of study falls within Smart Mobility, a Big Data Analytics sub-area, that makes cities infrastructure and public services more iterative, efficient and intelligent (Pellicer, S. et al., 2013). United Nations (United Nations, 2011) affirms that the world's urban population will increase from 2.6 billion in 2010 to 5.2 billion in 2050, representing 70% of world population. IPEA (IPEA, 2015) shows that 65% of Brazil population living in capitals use public transportation. The city of São Paulo - most populous Brazilian capital - rose by 14 million bus users in the first quarter of 2014, according to the SPTrans, enterprise responsible for transportation service management in São Paulo, representing an increase of 2.7% over the same period of 2013. Population growth will bring many challenges to cities, such as an increase in the number of public transport users. This can significantly affect the whole bus public service, as most of the time we do not know exactly how the number of passengers is distributed among the lines and if a certain area of the city is being supplied by the right quantity of buses. Therefore, improving quality and availability of public bus service has become essential. Passenger flow and density analysis provides great understanding of bus service network, showing its real demand and usage, allowing better control, management and decision-making.

(QING, 2009) use data from Beijing Smart Card to show that it is an important source of information of passengers’ behavior. The passenger flow analysis, combining GPS data and Smart Card is proposed by (DUAN, 2012). (KIEU, 2015) use Smart Card information to focus on passenger segmentation - or characterization. (ZHANG, 2014) use the same combination of data sources; however, aims to calculate the passenger density of a bus service, showing a model of buses schedule table redefinition. In the context of flow analysis and passenger density, none of those works has as focus usage of GPS data and smart card together with city areas clustering. (GUIDO, 2017) one of the bases for this work, presents a Decision Support System (DSS) framework for assisting transport operators and planners in making decisions regarding sustainable mobility development, as well as attracting car users to make them migrate to public transports. (BRIAND, 2017) presents a model which applies Gaussian mixture model to regroup passengers based on their temporal habits in public transportation usage. (ERATH, 2017) is a literature review which brings together recent advances in Big Data stack to understand travel behavior and inform travel demand models that allow to compute what-if scenarios. Moreover, this thesis is the first work in literature to implement Lambda Architecture (KIRAN, 2015) as the architecture to deal with this type of data.

The thesis is organized as: Section 2 presents related works, containing the state of the art of this scenario and important papers; Section 3 presents Lambda Architecture and LDAVI – Lambda Architecture Driven Implementation, the proposed method; Section 4 presents the case of study and its evaluation and Section 5 and 6 give the conclusion and the references, respectively.

2 RELATED WORK

Lambda Architecture is a quite new approach to handle Big Data problems. This architecture is capable of handling a huge amount of data, by having distributed and scalable storage infrastructure and a processing layer supporting both batch and stream modes.

Within the state-of-the-art of Lambda Architecture and urban mobility area of Smart Cities, mainly solutions and models for density and passenger flow analysis and Origin-Destination (OD) matrix computation were analyzed. Origin-Destination (OD) matrix is a matrix that provides passengers travelling information as where the journeys begin and end (NASIBOGLU, 2012). There are three main methods to solve this problem: intersection of GPS data and Smart Card; separation of the bus lines in different segments to determine user's shipping segment; and usage of buses timetables. The first uses time-matching algorithms, but may become impractical to require that buses have GPS and Fare Collection Systems. The second has low precision results, while the third can be affected by several factors such as bus speed and traffic.

This chapter presents a literature review approaching Lambda Architecture and the three methods described above and also architectures and techniques that surround this subject.

2.1 Literature Approaches

Firstly, we present three literatures describing the Lambda Architecture, and then those that focus on the same scope of this work. Lambda Architecture has two different data processing streams: batch processing – used for massive workloads for delayed processing – and stream processing – or real-time processing for fast data streams (GRIBAUDO , 2018). The architecture will be further explained in Section 3.

After, we describe literatures focusing on our use case scenario related to Smart Mobility. Some techniques and tools are similar between then, however important variations appear. In this section we describe, one-by-one, the most relevant works and in Section 2.2 a comparative analysis is presented containing the most relevant differences between them.

(KIRAN, 2015) combines lambda architecture approach with cloud development. It implements a lambda architecture design to build a data-handling backend supported by Amazon EC2. The idea behind the implementation is to provide a data management and processing framework to minimize network resources, cost and on-demand availability. Results show reduction in cost and benefits for performing online analysis and anomaly detection for sensor data. (VILLARI, 2014) approaches two main challenges: big data storage and analytics, and large-scale smart environments management. A software solution combining AllJoyn and lambda architecture is proposed, enabling big data storage, processing and real-time analytics. MongoDB - NoSQL database - and Apache Storm - distributed system for real-time processing of data streams - fitting in IoT smart environments, compose the solution. (GRIBAUDO, 2018) presents a modeling approach of Lambda Architecture which provides a fast evaluation tool to support design choices about parameters leading to better architecture designs. (MUNSHI, 2018) shows a smart grid big data eco-system based on Lambda Architecture state-of-the-art for batch and real-time operations. The features of using this type of architecture for Smart Grids are presented, such as robustness and fault tolerance, low latency, scalability generalization and flexibility. The eco-system uses Hadoop Big Data Lake. (YANG, 2017) implements RADStack, an open-source Lambda Architecture implementation, designed to provide fast, flexible queries and overcome limitations of pure batch processing or pure real-tile systems. It uses Apache Kafka, Apache Samza, Apache Hadoop and Druid.

In this work, we are presenting motivation for developing such architecture, how it works and how we implemented it. Lambda Architecture is comprised by three layers: batch, speed and serving. Thus far, batch layer is implemented employing Apache Hadoop, Apache Spark, Apache Kafka and Apache Zeppelin. We also briefly review the other two layers in order to implement them in the next phase of our work, where for serving and speed layer we conclude that Storm is the best choice.

Relate to Smart Mobility, (QING, 2009) only address the analysis of IC Card - Smart Card - without considering other data sources, to prove that it is a major source of information passenger traffic. It uses the attraction weight coefficient OD calculation for the database of public transport passenger flow.

(LI, 2012) address the use of the second and third methods by proposing two steps: separation date card and bus stops matching. It uses the idea of Automatic Data

Collection Systems (ADCS) and aims to describe a method to determine the original location by associating bus card data and spatial relationship of stop lines or destination.

(DUAN, 2012) proposes a method for integrating data originated from GPS and Smart Card, with the goal of analyzing passenger flow of single-line type for Beijing. The work analyzes passenger information from one bus route of Beijing, including passenger flow rates, flow from one station and flow of a line, through combining GPS and Smart Card data. (ZHANG, 2014) presents, for the first time, the joint use of data from GPS and Smart Card to calculate passenger density on a bus service. It proposes a new method for fusion of GPS data and Smart Card that can reduce the timing error in the type Fare Collection Devices (FCD) data. (KIEU, 2015) use Smart Card information to focus on passenger segmentation - or characterization.

(GUIDO, 2017) one of the bases for this work, presents a Decision Support System (DSS) framework for assisting transport operators and planners in making decisions regarding sustainable mobility development, as well as attracting car users to make them migrate to public transports. Data comes from C.O.R.E - Centrale Operativa REgionale - in Calabria, Italy, for acquiring transit data, and moreover, through data mining techniques, users' trip information including data such as origin and destination of passengers, routes and travel times was also obtained. The proposed framework is designed for collecting, integrating, aggregating, fusing, managing and exposing open Big Data, and is based on a centralized Database Management System (DBMS).

(BRIAND, 2017) presents a model which applies Gaussian mixture model to regroup passengers based on their temporal habits in public transportation usage. The case of study has been performed on five years of data collected in the city of Outaouiais (Canada), showing passenger clusters linked to their fare types and a relative stability of public transport usage. (ERATH, 2017) is a literature review which brings together recent advances in Big Data stack to understand travel behavior and inform travel demand models that allow to compute *what-if* scenarios. It describes many different categories of data used in the transportation are, such as mobile, smart card, GPS and point of interests (POI). Related to smart card data, it gives a deeper look in origin-destination matrices and activity identification.

2.2 Comparative Analysis

In this section, we firstly perform a comparative analysis between the five related works approaching Lambda Architecture. Through this analysis, we have a better understanding about the architecture itself as well as its applicable use cases. After, we perform another comparative analysis between all related works described in section 2.1 - discarding the five that focus on Lambda Architecture. This analysis considers four main items to highlight similarities and differences between them: purpose of the work, data source and structure and used methods and algorithms. The result of this analysis is described below and summarized in table *Table 1*. Sections 2.2.1 and 2.2.2 are strictly related to our use case of Smart Mobility.

2.2.1 Datasets

In the intelligent transportation scenario, literature provides different techniques and algorithms and data sources. For obvious reason, the two main data sources - datasets - used are the those containing GPS data and passengers’ behaviors - most coming from Smart Card. These two different datasets are usually crossed for having both passengers’ characteristics and OD matrix computation.

2.2.2 Algorithms

Time-matching (temporal) algorithm is the most used one to approach this type of challenge. This type of algorithm belongs to graph theory area and in this scope is used to correlate temporal and spatial metrics. (ZHANG, 2014) and (GUIDO, 2017) use this method to correlate public transport geolocation - GPS - and passengers origin-destination matrix. This approach is used in this work inside lambda architecture. Based on sub-section 2.1 other techniques and algorithms appear, such as attraction weight coefficient OD calculation and density-based spatial clustering of application with noise (DBSCAN). The goal of this algorithm is to identify clusters of high density and noise of low density - in this scope, a noise represents a trip randomly made, as well as to identify a cluster of any shape and size - which can truly represent a human behavior pattern regarding trips. A strong benefit of this algorithm is that is does not require predetermination of initial cores of number of clusters. This method is used by (KIEU, 2015) but is not the case of our work as we have clusters for representing bus-stops and city zones well determined.

2.2.3 Comparative table

We approach, for the first time in literature, a Lambda Architecture implementation - built with Apache Hadoop and Apache Spark - focused on computing passenger and bus data, providing parallel and distributed storage and processing. (GUIDO, 2017) is taken as one of our bases, however we go further providing bus-stops and city zones categorization. Thereto, not just single journeys are considered but also connection trips and round-trips, which provide a full understanding of passengers flow.

Table 1 shows the main contributions, technologies and applications of Lambda Architecture within the five related works.

Table 1 – Lambda Architecture literature

	Contribution	Technologies and Application
KIRAN, 2015	Lambda architecture design to build a data-handling backend supported by Amazon EC2. Goal is to provide a data management and processing framework to minimize network resources, cost and on-demand availability.	Cloud computing, used by Amazon EC2
VILLARI, 2014	Software solution combining AllJoyn and lambda architecture is proposed, enabling big data storage, processing and real-time analytics.	MongoDB and Apache Storm - distributed system for real-time processing of data streams - fitting in IoT smart environments.
GRIBAUDO, 2018	Modeling approach of Lambda Architecture that provides a fast evaluation tool to support design choices about parameters leading to better architecture designs.	N/A
MUNSHI, 2018	Lambda architecture implementation for Smart Grids. The features of using this type of architecture for Smart Grids are presented, such as robustness and fault tolerance, low latency, scalability generalization and flexibility.	Hadoop Big Data Lake, Flume, Apache Spark, Apache Hive, Apache Impala, Tableau
YANG, 2017	RADStack, an open-source Lambda Architecture implementation, designed to provide fast, flexible queries and overcome limitations of pure batch processing or pure real-tile systems.	Apache Kafka, Apache Samza, Apache Hadoop and Druid.

Regarding the use case approached in this work, *Table 2* shows the main contributions, methods and dataset used by the six related works described above focusing on Smart Mobility.

Table 2 – Smart Mobility literature

	Contribution	Method	Dataset
QING, 2009	Prove that IC Card is an important source of knowledge about the passenger flow.	Attraction Weight Coefficient OD Calculation	N/A
LI, 2012	Method to determine the original location through the Smart Card data association and the spatial relation between bus stations	Schedule tables and lines segmentation methods	Smart Card GPS of Beijing
DUAN, 2012	Characterize flow and OD matrix of one bus line from Beijing	Combine GPS and Smart Card data	Smart Card GPS of Beijing
ZHANG, 2014	Passenger density and flow computation. Evaluation of buses schedule tables choices	Time-Matching algorithm	Smart Card GPS of Shenzhen
KIEU, 2015	Passenger segmentation using smart card data and DBSCAN algorithm	Density-based spatial clustering of application with noise (DBSCAN) algorithm	Smart Card
GUIDO, 2017	Decision Support System (DSS) framework for public transport decision-making	Spatial-temporal matching algorithm	Transit data
BRIAND, 2017	Model which applies Gaussian mixture model to regroup passengers based on their temporal habits in public transportation usage	Gaussian mixture model	Smart Card
ERATH, 2017	Literature review which brings recent advances in Big Data stack to understand travel behavior and inform travel demand models that allow to compute what-if scenarios. It describes many different categories of data used in the transportation are, such as mobile, smart card and GPS. Related to smart card data, it gives a deeper look in origin-destination matrices and activity identification.	N/A	Smart Card, GPS

3 LAMBDA ARCHITECTURE

Lambda Architecture (KIRAN, 2015) is a new architectural concept that emerged with the raise of Big Data Analytics. It was first proposed by Nathan Marz (MARZ, 2013) in 2013 based on hist experience working on distributed data process at Twitter, and came to satisfy the needs of a robust system – generic, scalable and fault tolerant data processing architecture. The architecture is mainly composed by three different layers, responsible for handling massive amount of data. Batch layer – for historical data processing - serving layer and speed layer – for streaming processing (GRIBAUDO , 2018).

3.1 State-of-Art

In the state-of-art, three different layers compose this architecture pattern: batch layer, speed layer and serving layer. The first one – batch layer – is responsible from processing all available data using a distributed processing system - batch computation. This layer may be useful when we deal with non-live data, monthly reporting, user behavior, or heavy processing that cannot be performed on live. The second, speed layer, processes data streams in real time and provide real-time views of recent data – this type of processing is also known as stream processing. This layer may be useful for real-time processing, where we deal with critical problems – airport and flight control, ticket sale, healthy, social medial. The third one, serving layer, is responsible for merging the outputs from the batch and speed layers, responding to ad-hoc queries. Literature says that for implementing Lambda Architecture there is no need to implement the three layers, but one can models it and uses only one or more combined, depending on the problem to be solved. According to Nathan Marz (MARZ, 2013) some of other motivations for building a Lambda Architecture are robustness and fault tolerance – hardware failures and humans failures - scalability, generality and extensibility – features can be added easily and be easily maintained. *Figure 1* shows the Lambda Architecture. This architecture is already in place in some important market leaders such as Google, Amazon, Twitter, Amadeus, Netflix and Samsung.

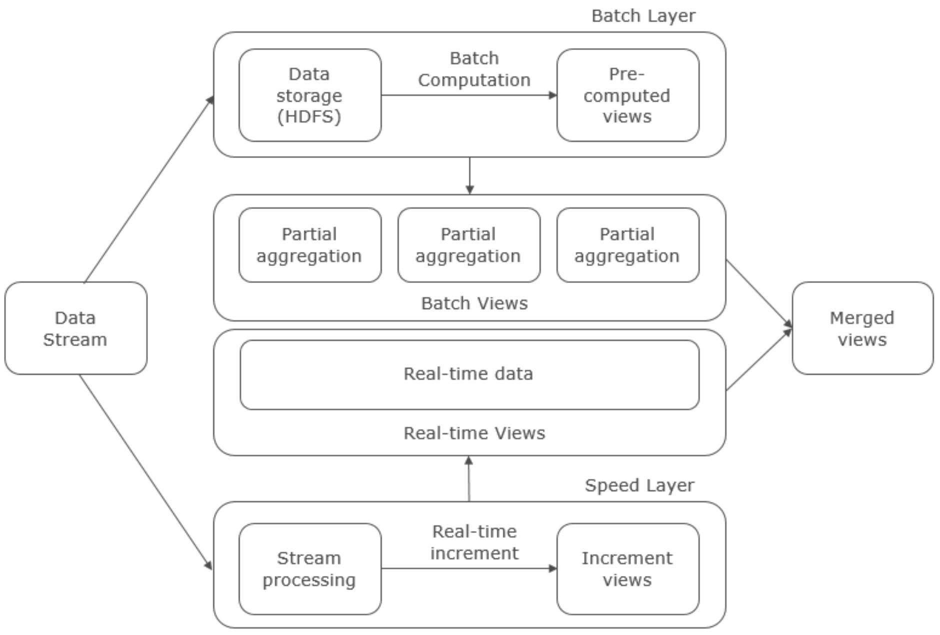


Figure 1 – Lambda Architecture

3.2 Technology Stack

The stack for building a Lambda Architecture is vast and it can be defined per layer. Starting with distribution and storage, Apache Kafka, Avro, Hadoop HDFS, Cloudera Impala – native analytic database for Apache Hadoop, which translates parquets files to SQL-like structure - Apache Hive, Elasticsearch and Apache Cassandra are often the chosen technologies. Both stream and batch layers can take advantage of them, however is mostly batch layer that uses storage technologies as it is the one responsible for computing historical data, while stream layer tends to do all computation online and only store the real-time views. HDFS is a distributed file system that stores data on commodity machines and, as advantage, replicates data avoiding data loss. It is not a database, but a data storage that can be used by several data warehouses or databases – such as Apache Hive, Apache Cassandra and Impala. HDFS also has the possibility to replicate data, reducing data loss and increasing fault-tolerance. Moreover, data can be stored in many ways and formats inside HDFS, and it has its own internal computation to compress data for reducing data size. For computation, we can highlight Apache Hadoop, Apache Spark and Spark Streaming, Apache Drill and Apache Storm and HBase. Spark has come taking the place of Hadoop MapReduce as it has often up to 100x better performance when running in memory and 10x better when running on disk. The reason is because Spark loads all data in memory, which sometimes may also be a limitation if infrastructure has not enough memory to enable Spark to do it. Spark Streaming is used for building the stream layer. In terms of data visualization and analytics, Apache Zeppelin, Tableau, R and Power BI appear as the most used technologies.

4 LDAVI – LAMBDA ARCHITECTURE DRIVEN IMPLEMENTATION

Lambda Architecture Driven Implementation (LDAVI) is a generic framework based on the Lambda Architecture and built using Big Data stack, such as Apache Hadoop, Apache Spark and Kafka. LDAVI is an implementation of Lambda Architecture (KIRAN, 2015) using Big Data. In this work we approach and provide an implementation of this architecture, building it with the main Big Data technology stack. Although it has started being used in some areas such as search engines and platforms requiring real-time processing – such as video stream players – we demonstrate that this architecture can also bring benefits for Smart Mobility, more precisely in public transportation. Differently from related works, we approach three different types of trip: simple trip, connection trip and round trip, what makes the analysis complete and more accurate.

We perform a use case based on Smart Mobility to demonstrate its feasibility and performance capacity. It uses bus information such as data from buses GPS, bus stop geolocation and passengers smart cards and through a big data architecture and methods, processes it, extracting useful information such as passengers' density and flow, and bus stops segmentation based on travel purposes. Results obtained from the processing through our Lambda implementation show the real bus service demand, allowing a better understanding about the bus service network and being an important decision-making source for authorities and responsible for improving this type of service. Moreover, among the benefits of applying this implementation we find balanced latency, throughput, scalability and fault tolerance.

4.1 Overview

This work aims to build a generic architecture – based on Lambda Architecture. The main goal of the framework is to provide a clear deep analysis in the architecture itself – literature - as well as in its implementation, describing the used technologies. Applying this Big Data architecture implementation allow us to execute both batch processing – historical data – and stream processing – real-time analysis due to the implementation of the architecture's layers, built on top of Big Data stack technologies described in section 3.2.

To demonstrate the operational capacity of the proposed architecture, this work applies it in Smart Mobility area. It gives a better understanding about density and flow of bus passengers by performing in-depth analysis in passengers' smart cards and bus geolocation. In our approach, as we are mainly interested in understanding density and flow of passengers, we focus our experiments on Chapter 4 in historical data computation – batch processing. Nevertheless, future work can include stream processing in case we want to have a real-time computation, which could allow us to compute arrival time of buses or even live demand and flow among the whole bus network.

A density-and-flow-map of a bus line is built representing the real need and usage of buses, allowing decision-making for buses management and control. Results of the bus stops and city classification can be modeled, according to a defined precision radius representing the relation between the bus position and the bus stops location. Defining, for instance, that each bus stops owns a radius of 50 meters from its real location means all GPS buses records that have a position in this radius precision belong to that bus stop. The model also builds a city characterization, aiming to categorize it in a scale of purposes areas types: residential, work/business, nightlife, and areas with small or high passengers' density and flow. These categories are pre-defined and chosen to be the most general as possible, representing the main city activities, without losing information. Future work can improve this categorization and find more purposes by using other clustering techniques.

The travel proposes criteria are pre-defined according to the daytime that the buses are taken. In this thesis, we segment travel purposes among *Residential, Night Life* and *Work.*

4.2 Considerations

Table 3 shows the main contributions, technologies and applications of Lambda Architecture within the five related works and this work.

Table 1 - Lambda Architecture literature

	Contribution	Technologies and Application
KIRAN, 2015	Lambda architecture design to build a data-handling backend supported by Amazon EC2. Goal is to provide a data management and processing framework to minimize network resources, cost and on-demand availability.	Cloud computing, used by Amazon EC2
VILLARI, 2014	Software solution combining AllJoyn and lambda architecture is proposed, enabling big data storage, processing and real-time analytics.	MongoDB and Apache Storm - distributed system for real-time processing of data streams - fitting in IoT smart environments.
GRIBAUDO, 2018	Modeling approach of Lambda Architecture that provides a fast evaluation tool to support design choices about parameters leading to better architecture designs.	N/A
MUNSHI, 2018	Lambda architecture implementation for Smart Grids. The features of using this type of architecture for Smart Grids are presented, such as robustness and fault tolerance, low latency, scalability generalization and flexibility.	Hadoop Big Data Lake, Flume, Apache Spark, Apache Hive, Apache Impala, Tableau
YANG, 2017	RADStack, an open-source Lambda Architecture implementation, designed to provide fast, flexible queries and overcome limitations of pure batch processing or pure real-tile systems.	Apache Kafka, Apache Samza, Apache Hadoop and Druid.
This work	Implement a low-cost framework based on Lambda Architecture and Apache technologies	Apache Hadoop, Apache Spark, Apache Kafka, Apache Impala, Scala

Regarding the use case approached in this work, Table 2 shows the main contributions, methods and dataset used by the eight related works described above focusing on Smart Mobility.

Table 2 - Smart Mobility literature

	Contribution	Method	Dataset
QING, 2009	Prove that IC Card is an important source of knowledge about the passenger flow.	Attraction Weight Coefficient OD Calculation	N/A
LI, 2012	Method to determine the original location through the Smart Card data association and the spatial relation between bus stations	Schedule tables and lines segmentation methods	Smart Card GPS of Beijing

DUAN, 2012	Characterize flow and OD matrix of one bus line from Beijing	Combine GPS and Smart Card data	Smart Card GPS of Beijing
ZHANG, 2014	Passenger density and flow computation. Evaluation of buses schedule tables choices	Time-Matching algorithm	Smart Card GPS of Shenzhen
KIEU, 2015	Passenger segmentation using smart card data and DBSCAN algorithm	Density-based spatial clustering of application with noise (DBSCAN) algorithm	Smart Card
GUIDO, 2017	Decision Support System (DSS) framework for public transport decision-making	Spatial-temporal matching algorithm	Transit data
BRIAND, 2017	Model which applies Gaussian mixture model to regroup passengers based on their temporal habits in public transportation usage	Gaussian mixture model	Smart Card
ERATH, 2017	Literature review which brings recent advances in Big Data stack to understand travel behavior and inform travel demand models that allow to compute what-if scenarios. It describes many different categories of data used in the transportation are, such as mobile, smart card and GPS. Related to smart card data, it gives a deeper look in origin-destination matrices and activity identification.	N/A	Smart Card, GPS
This work	Implement a low-cost framework based on Lambda Architecture for bus service management, aiming to extract passengers density and flow through Smart Card, bus stop geolocation and buses GPS data	Time-matching algorithm applied over GBS, Smart Card and bus stops geolocation data in a Lambda Architecture	Smart Card, Bus GPS and bus stops geolocation of Shenzhen

Differently from relate works presented in the two tables above, our work combine a Lambda Architecture implementation, using the main Big Data stack – Apache stack – with Smart Mobility area. A data processing framework has been built on top of Apache Kafka (message broker), Apache Hadoop and Apache Impala (storage), Apache Spark (data processing) and Scala (functional programming language), and Apache Zepellin (data visualization). This shows the feasibility and powerfulness of Lambda Architecture, and by our case of study in section 4, and, among the related works found in literature, for the first time apply it to a Smart Mobility problem.

4.3 Architecture

Lambda Architecture Driven Implementation (LDAVI) is an implementation of Lambda Architecture (KIRAN, 2015). From section 3.1 where we presented the Lambda Architecture – *Figure 1* – what we have implemented is the batch and the serving layer – merged views from batch layer. Those 2 layers allow us to compute historical data and adding stream layer is not complex as it uses the same platform of the batch layer, except that the logic behind is different. Bellow in *Figure 2* the highlighted part is what represent both batch and serving layers.

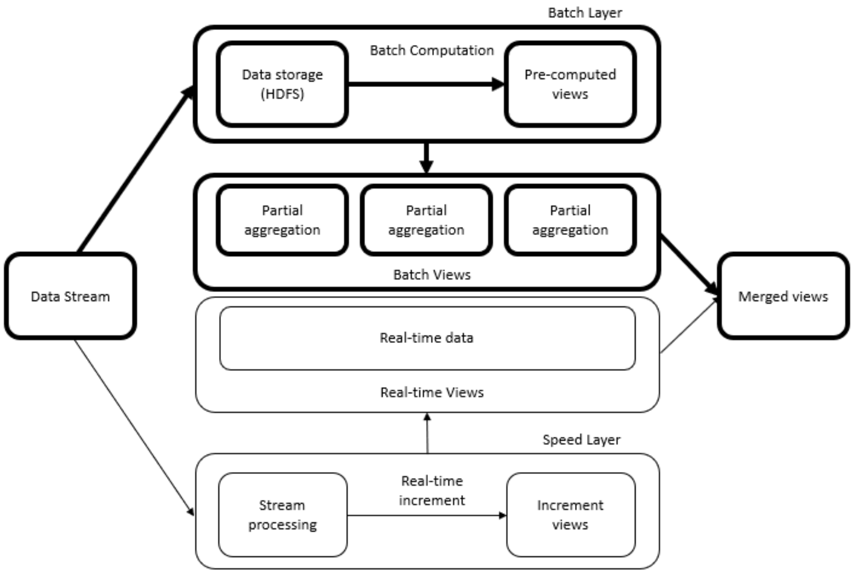


Figure 2 - LDAVI implementation: batch and serving layers

4.3.1 Data Flow

The data flow within the architecture can be represented by three different phases: data ingestion, data processing and data visualization – *Figure 3*. This represents what is explained in subsections 4.2.1.1, 4.2.1.2 and 4.2.1.3.

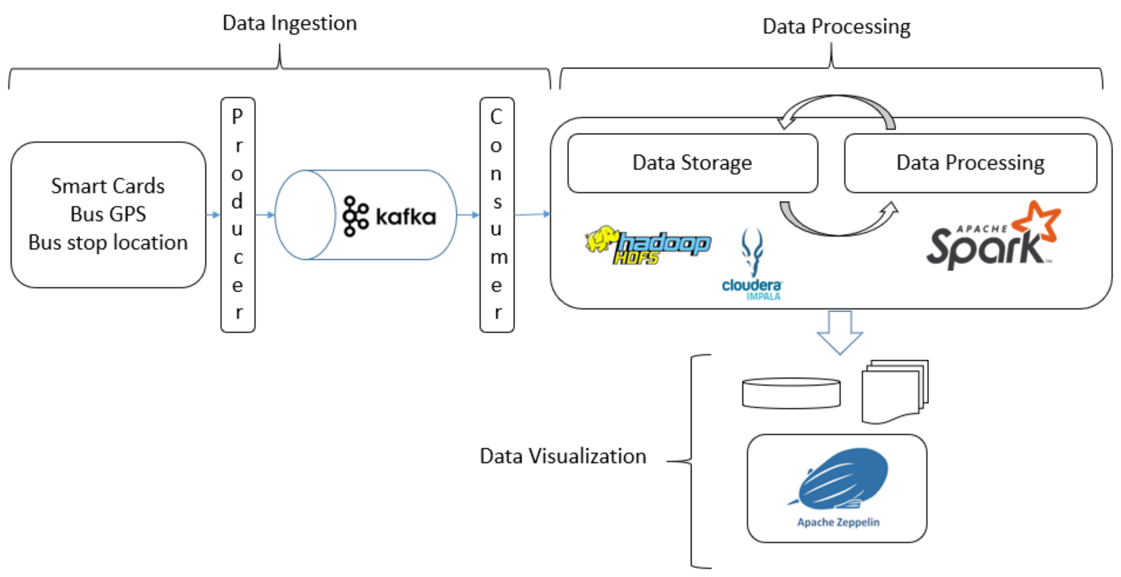


Figure 3 - Data ingestion, data processing and data visualization phases

All of them can be implemented in different ways using several different frameworks and technologies. These three phases as well as the technologies used in this thesis to implement them are further described in the next subsections.

4.3.1.1 Ingestion

This step of the data flow within the architecture is responsible for receiving the data coming from one or several sources. In our case, data comes from bus GPS and passengers Smart Card data.

The core of this step is Apache Kafka, a message broker built using producer-consumer approach. Kafka's advantages include the possibility of having different sources of information pushing data to it. Furthermore, it relies on partitions and topics, which means we can clearly separate different data coming from different sources in different topics- *Figure 2*. Topics represent categories of messages to which records are published to and are composed of partitions, representing an ordered and immutable sequence of records. Kafka allows topics and partitions configuration relying on data replication and fault tolerance. Producers and consumers can read and write data from one or several topics and can use the partitions to enable data redundancy. A producer has been built for getting data from our sources and pushing it into Kafka, and, respectively, a consumer has been built for reading this data from Kafka and ingesting into Hadoop Distributed File System - HDFS. Hadoop comes from Google's MapReduce and Google File System, (Big Data: Hadoop, Business Analytics and Beyond). Main components of Hadoop are: HDFS, default storage layer in a given Hadoop cluster; Name Node, node in Hadoop cluster that provides the client information on where in the cluster particular data is stored and if any nodes fail; Secondary Node, backup of the Name Node; Job Tracker, node in Hadoop cluster that initiates and coordinates MapReduce jobs or the processing of the data; Slave Nodes, store data and take direction to process it from the Job Tracker.

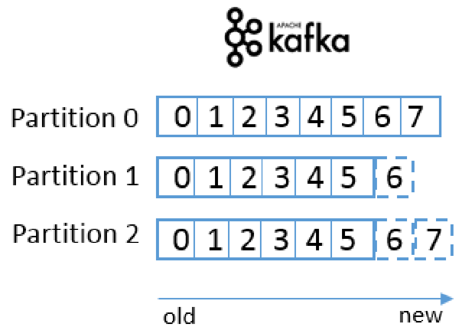


Figure 4 – Kafka topic structure

The format for data storage chosen in this thesis is parquet files, which rely on Apache Parquet, a columnar storage format available to any project in Hadoop ecosystem. On top of our data store layer, we use Apache Impala, a native analytic database for Apache Hadoop, which translates parquet files in SQL-like structure.

4.3.1.2 Processing

The second phase is the data processing, responsible for accessing and processing the information stored into HDFS. This can be performed in several ways, and the two most known for processing data in Big Data is MapReduce and Apache Spark – general engine for large-scale data processing. MapReduce was the first approach used in the literature, however it presents some limitations comparing to Apache Spark. Spark can have up to a 100 times better performance when running in memory and up to 10 times when running on disk. This main difference is due to the way both handle data. Meanwhile a significant part of MapReduce running cost comes from reading and writing operations on disk, Spark loads all data in memory. It is important to highlight the fact that the choice must be done regarding the memory and disk spaces (GU, 2013) (ZAHARIA, 2012).

In this thesis, our data processing layer uses Spark jobs, which have been implemented in Java and Scala and are responsible for aggregating data stored in Hadoop, extracting passenger density and flow. Apache Oozie is also used for scheduling the jobs. Spark has four libraries: Spark SQL, for formatting and structuring data and enabling query requests; Spark Streaming, for real-time and streaming applications; MLib, for machine learning approaches; GraphX, for graphs and graph-parallel computation. In this work, we use only Spark SQL for querying Apache Impala.

4.3.1.3 Visualization

The last phase is the data visualization, responsible for exposing the results processed by Spark jobs. There are many tools to visualize data in Big Data area. The choice of how we expose and display data for the target users is extremely important, as is through this visualization that knowledge will be extracted and decisions will be taken. One of the tools that has gained strength in the past years is Tableau Software, a commercial tool having an in-memory data engine and live query engine presenting a fast performance when handling massive data. As one of our goals was to implement a low cost framework, mainly based in Apache ecosystem, the tool used to expose our passengers density and flow was Apache Zeppelin, a web-based tool for dashboard generation (Streaming Data Analysis using Apache Cassandra and Zeppelin, 2016). It enables data-driven and supports Apache Spark and Python.

Summarizing all our architecture layers, a Kafka producer is responsible for ingesting data into a Kafka topic and a Kafka consumer reads it and stores into HDFS. Apache Spark jobs are responsible for processing and aggregating data. Passengers flow and density are computed using algorithms that will be further explained. Results - flow and density – are exposed through dashboards, using Apache Zeppelin. These phases are shown in *Figure 3*.

In the top of our implemented Lambda Architecture, we have applied some algorithms to address bus management area. The processing part can be divided into two parts, where the first has as objective to find passenger's density and flow, and in the second one we use this extracted knowledge to segment bus stops and areas of the city based on travel purpose. They are described as knowledge retrieval and segmentation, respectively.

4.4 Data Analysis

Data analysis represents the information retrieval – computing origin-destination matrices to find passengers' density and flow – and data segmentation – categorizing bus stops and city zones accordingly to travel purposes. This part of the process represents a proof of value of our Lambda Architecture framework.

4.4.1 Knowledge Retrieval

This first computation step is in charge of computing the input passengers' information – in our case, smartcards, buses GPS and buses stops geolocation – retrieving their density and flow among the bus network. As we are interested in computing historical data, we use the batch layer of our proposed architecture to build origin-destination matrices of passengers and then to apply the proposed algorithms on top of them. Batch layer, as mentioned, could be used for many different types of computation and knowledge retrieval in case of dealing with historical data, as well as the speed layer could be used for real-time analysis purposes. Kafka producer feeds Kafka with the data and another consumer consumes from it, storing data in HDFS for backing up, reprocessing needs and fault-tolerance purposes. Therefore, as soon as data arrives in HDFS it is taken by another Spark job to be part of our ETL – Extract, Transform, Load – process, which applies the necessary algorithms to retrieve the O-D matrices. *Figure 3* illustrates this process. The goal in this thesis is also being as much generic as possible, and further experiments are described in section *5 Experiments*. Following subsections describe how to use Lambda Architecture batch layer to compute origin-destination matrices.

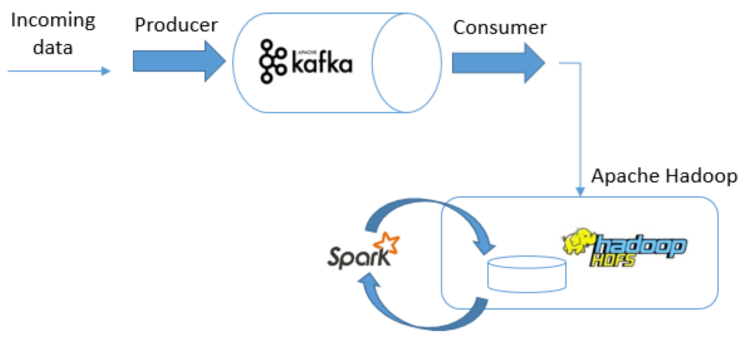


Figure 5 – Lambda architecture batch layer

4.4.1.1 Density and Flow Computation

Having smartcards, bus GPS and bus stops geolocation, we are able to start the computation process. It means that we want to figure out the passenger's density and the passengers flow in a determined bus line, and, in a future work, in the whole bus public service network. GPS dataset has buses latitude and longitude coordinates, Smart Card dataset has tapping card records of all buses users and bus stops geolocation is a static dataset containing the position - coordinates - of all stops of the used bus. The three datasets are be combined to get the OD matrix for each bus passenger. For passenger origin derivation process we must be aware that the time recorded in a tapping card stored into Smart Card dataset represents the moment when the passenger boarded. In comparison with other works in the literature (ZHANG, 2014) we also make use of time-matching method to build the OD passengers matrices. Once this computation is done, e. g, all the passengers origins and destinations are mapped, it is possible to make inferences and to analysis the density and the flow, obtaining source of knowledge to segment the bus stops and city using clustering algorithms.

First, the goal is to build the Origin-Destination matrices. Having the coordinates - latitude and longitude - of each smart card record and bus stops, we can identify in what bus stop the passenger boarded. We assume passenger taps the card as soon as he gets on the bus and that all passengers boarding are paying the fare through Smart Card usage. Data pre-processing is necessary before starting the computation process. The GPS data and Smart Card data must be separated according to the target bus line to focus the computation process separately for all buses of each bus line.

The boarding station is the easy one to be obtained, meanwhile for the alighting station some assumptions are necessary, as the passenger does not tap the card when he gets out of the bus. Following the state-of-art of OD matrix computation in public transport area (DUAN, 2012), (ZHANG, 2014) this work will make use of time-matching algorithm. The next steps are the Smart Card ID and GPS ID pairs discovery, which represents the respective smart card record related to a GPS record. Thereunto, we must compare the boarding station time and the tapping card time. Through this comparison and considering the minimum time difference between GPS record and Smart card record, we are able to find out the passenger boarding station. GPS records that have their coordinates positioned inside a bus stop coverage radius belong to it. From that, we can pre-process and determine, for each bus position, if this bus is in a bus stops coverage radius. Then, the problem is limited to compare GPS and Smart Card times using time-matching algorithm.

The process of obtaining the passenger destination requires some assumptions and it is an estimation process since we do not have smart card records for passengers alighting the buses. To allow this computation, time slots are determined and classified according to travel purposes. Only those that can be obtained analyzing passenger density in a period of time were used. Furthermore, they represent basic travel goals as going to work or going to home, or even going to a party or a pub at night. There is not possible to analyze if a passenger is taking a bus to go to a mall or to a supermarket, for example, since it is an activity that can be done any time in a day. This is the first step of bus stops and city segmentation that will be deeper discussed in the next section. As described in section 3.1 this work considers three different segmentations: work/commercial, residential and nightlife, which have been arbitrary defined by the author. Passenger's density represents the density – number – of passengers inside the bus in each stop, which is incrementally computed by considering the number of passengers that have boarded in the bus minus the ones that have gotten off.

For estimating passenger destination, two different scenarios must be considered so we can understand if it is a round-trip, a connection trip or a simple trip. In the first scenario is the passenger taps the card just once and falls into a simple trip purpose. (ZHANG, 2014) approach only this case and makes all the computation based on that. Here, moreover considering a simple trip, we also take into account the case when the passenger taps the card twice or more, what can be done when he is performing a connection trip or a round-trip. For estimating the destination of a simple trip, we must know the most common passengers destinations based on the time slot of the day. In other hand, if the passenger taps the card twice in a short period of time it falls into one of these two cases: connection trip and round-trip. If the time difference between two taps is below 60 minutes, we determine it is a connection trip. If the time is over 60 minutes and below 120 minutes, it is a round-trip. This estimative is performed in cascade mode, which means that if the passenger taps the card three times or more, we determine the trip type and the destinations one by one always analysis the previous tap. We must know that the goal is to find out the passenger final destination and the trip type - simple trip, connection trip or round-trip. The *Figure 3* below shows that estimative:

Table 3 – Trip type classification

Trip Type	**Taps**	**Difference between taps**	**Estimation Method**
Simple	1	-	-
Connection	2 or more	< 60 minutes	First trip destination is the second trip origin. Second trip destination estimated as a simple trip.
Round	2 or more	60min < d < 120min	First trip destination is the second trip origin. Second trip destination is the first trip origin.

4.4.1.2 O-D Matrix Computation

Origin-destination matrix shows the travel path done by each passenger and is used to calculate the density and flow of the bus line. Datasets have information allowing the origin discovery, as the Smart Card records have the timestamp of when the passenger taps the card. Following our assumption, it is equivalent of boarding time. As explained in section 3.4, each Smart Card record will be paired to a GPS record and through that, there is enough information for finding in which station the passenger has boarded.

For finding the destination, first we should know the passenger most common destinations - as home, work and nightlife. Knowing that allow us to know the bus stops the passenger usually uses as origin and as destination. The algorithms for finding out origin and destination, inspired in (ZHANG, 2014) are shown below - Algorithm 1 and Algorithm 2 - where the tapping time is represented by x:

```
Algorithm 1: Finding passenger's origin
1: For each Smart Card record, find the GPS record with the minimum difference of time between tapping and arriving
2: The result is the most likely vehicle that the passenger gets on
3: Store that Smart Card and GPS pair
4: Repeat the process for all Smart Card records
```

```
Algorithm 2: Estimating passenger's destination

1: If it is between Monday and Saturday
2:     switch (x)
3:         case 6am <x < 8am
               Trip purpose may be residential
4:         case 5pm <x < 7pm
               Trip purpose may be work
5:         case 11pm <x < 5am
               Trip purpose may be nightlife
6:         default
               Trip purpose may be personal
7: else
8:     switch (x)
9:         case 11pm <x < 5am
               Trip purpose may be nightlife
10:        default
               Trip purpose may be personal
```

After processing all passengers' data, the result of the algorithms – density and computation – is again stored in Hadoop Filesystem for being re-used in the segmentation phase.

4.4.2 Segmentation

As the information retrieval process described in section 3.3, another Spark job takes care of segmenting the data – passengers' flow and density – based on travel purposes. This is also performed in the batch layer of our Lambda Architecture implementation. In most of cases described in literature, our segmentation process is done in off-line mode – not needing real-time analysis. Clustering algorithms could have been used in this phase, however the goal of this thesis is to implement Lambda Architecture and prove our generic architecture can be applied in bus services management area. Therefore, after having all origins and destinations matrices, bus stops and city zones are segmented taking into account the pre-defined time slots representing travel purposes. It is done by considering in which time slot the Smart Card tap timestamp fits in. Each bus stop is categorized considering the trips purposes of the passengers boarding and alighting (*Table 4*). The direction of the bus must be considered - incoming or outcoming buses. The time slots were built, for generalization, even we know some cities - especially metropolis – use to present discrepancy when compared to the slots – when having 24h industries and stores, or with an intense nightlife. Income Time Interval and Outcome Time Interval represent the time of buses arriving, or leaving, respectively, a city zone or bus stop.

Table 4 – Trip type behaviour classification

Segment Name	**Income Time Interval**	**Outcome Time Interval**
Residential	6pm-8pm; 2am-6am	6am-8am
Work	7am-9am	5pm-7pm
Night Life	8pm-11pm	11pm-5am

It is important to highlight that this categorization is not restricted to just one category, but to as many as it can have. For instance, a station can be most used for getting home purpose - 50% of the time - but is also responsible for an intense nightlife - 40% - being used in a short amount of time for personal purposes - 10%. We further describe the segmentation in section 4 during our experiments.

5 EXPERIMENTS

This chapter contains experiments that aim to demonstrate the feasibility of applying a Lambda Architecture based framework in smart transportation area. Experiments demonstrate the usability and feasibility of our implementation built using main Big Data stack. Here we focus on Smart Mobility problem – bus passengers density – but the scope can be extended to others Big Data challenges, as the technologies are the same. In our case, we use the framework for batch processing, as we want to detect the density and flow of passengers and to classify zones, which relies on historical data. Nevertheless, future work can make use of the same framework for stream processing, as the main pieces of the architecture would remain the same: Apache Kafka – message broker – Apache Hadoop – data storage - and Apache Spark – data processing.

First step is to check the data availability and how we can cross the different datasets to approach our target scenario. For each Smart Card record - representing a passenger boarding or landing - the matching algorithm is applied to find out its related bus stop. Further, during the classification phase, travel purpose is also found out. Bus stops and city zones are classified based on the main passenger's travel purposes leaving the bus stop. Results clearly give an important knowledge about the real bus service demand, which may be used by authorities to improve quality, efficiency and comfort. Again, this main propose of this work is to debate about Lambda Architecture and demonstrate how it can be used to handle Big Data problems by implementing a framework based on that.

In the next subsections, we describe the datasets, experiment methodology and evaluation comparing to our baseline. We divided our experiments into two different topics to differentiate what is the related to the architecture itself, contribution of this work, and what is related to our use case, representing the applicability of the architecture. First, we present technical experiments – dataset, methodology and metrics – and then data analysis experiments.

5.1 Technical Experiments

In this chapter, we describe technical aspects of our experiments such as dataset structure and methodology, which includes data processing and origin-destination matrix computation.

5.1.1 Experiments Set-up

In this section, we explain the set-up environment of the performed experiments. Datasets used as input data, methodology and metrics are described. In the next two chapters, the results of these experiments are exposed and analyzed.

As mentioned before, our Lambda Architecture implementation can handle a huge amount of data. To demonstrate a bit of its scalability and great performance, we apply our set-up to four different amounts of data, using the same dataset randomly replicated: 400Mb – original dataset; 4Gb, 20Gb and 100Gb. The goal is to demonstrate that as dataset grows, our architecture presents almost the same performance. Furthermore, we also change the backend infrastructure by adding Spark Workers node, which may affect the final performance. Workers are running Spark instances where executors live to execute tasks. They are the compute nodes in SparkThis architecture can be applied to many different use cases and even if for the one approach in this work a huge processing capability is not necessary, can be for other scenarios. For instance, the same experiment we present here for one bus line could be done in the whole bus network or could be combined with any type of traffic information.

Different set-ups used in the experiments are described below in table *Table 7*. We have eight different setups composing (four) 4 different experiments. Two (2) of them using 400Mb dataset, two (2) using 4Gb dataset, three (3) using 20Gb and one (1) using 100Gb dataset. The number of Spark Workers as well as their memories are also considered. As we do not want to focus in data recovery and replication, we set all partitions to five (5) and replication factor to one (1).

Table 5 - set-ups considering dataset size, Spark Workers and their memory and total available memory

Experiment	Data	Spark Workers	Cores	Spark Workers Memory
1	400Mb	1	8	6.8Gb
2	400Mb	3	8	6.8Gb
3	4Gb	1	8	6.8Gb
4	4Gb	3	8	6.8Gb
5	20Gb	1	1	8Gb
6	20Gb	1	8	6.8Gb

7	20Gb	3	12	8Gb
8	100Gb	3	12	8Gb

5.1.2 Data

For approaching our problem, we need three (3) types of datasets: data coming from passengers smart cards and geolocation of buses – for retrieving O-D matrices; and geolocation of the bus stops - for classifying stations and city. In our main experiment, those three different datasets are: GPS dataset, buses tops geolocation dataset and Smart Cards dataset. GPS dataset (*Bus ID, Time, Bus Lines, Latitude and Longitude*) contains 19485 records representing buses geolocation. Buses stops geolocation dataset (*Stop ID, Stop Code, Stop Name, Latitude, Longitude*) with all bus stops locations. Smart Cards dataset (*Smartcard ID, Time, Transaction Type and Metro Station/Bus Line)* contains Fare Collection System data and is composed by 3186 records representing passengers boarding and landing. Bus GPS and Smart Card datasets were obtained from (Schenzhen City in China), between October 22th, 2013 - 6am - and October 23th, 2013 - 12am, totalizing 400Mb. It is important to highlight the fact that although our framework is a Big Data framework with potential for handling huge amount of data, this first experiment aims to demonstrate that it can be used in the scope of smart transportation. After, we replicate this data up to 4Gb, 20Gb and 100Gb. As that, we can have a view about the scalability of our architecture. We could not increase even more due to hardware capacity limitation.

In GPS data, *Bus ID* represents the vehicle ID, Time is the record time, *Bus Lines* has the line of the recorded bus and *latitude* and *longitude* represent the position information of the vehicle. In the Smart Card data, *Smartcard ID* represents the ID of the tapped card, *Time* represents the moment the tapping information was recorded, *Transaction Type* - in our case - indicates that is a boarding, and *Bus Line* has the bus line - the Fare Collection System where the smart card data was recorded. Bus stop dataset can improve the accuracy and permit us to obtain better results once we know exactly the bus stops where the passenger boarded. *Bus Stop ID* represents the station id, *Latitude* and *Longitude* represent the bus stop position and *Terminal* indicates if the recorded bus stops is a bus terminal with connections. Passenger origin derivation process is considerably easy since we consider that the time recorded in a tapping card stored Smart Card dataset represents the moment when the passenger has boarded. For that propose, attributes as *SmartCard Id, Time* and *Bus Line* from a card record are used. In comparison with literature (ZHANG, 2014), this one also makes use of *time-matching* method to build the Origin-Destination (OD) passengers matrices. Once this computation is done, it is possible to make inferences and to analyze the density and the flow of all passengers that use the public bus service, obtaining source of knowledge to segment the bus stops and city using clustering algorithms. Having this segmentation will allow authorities to better understand the passengers trips purposes, giving a better decision-making source of information.

Table 6 shows GPS dataset structure, where *Bus ID* represents the ID of the vehicle, *Time* represents the time when recorded, *Bus Lines* has the line of the recorded bus and *latitude* and *longitude* represent the position information - coordinates - of the vehicle when recorded.

Table 6 – GPS dataset structure

Bus ID	Time	Bus Lines	Latitude	Longitude
55640	2013-10-22 09:45:56	03950	22.538700	113.972244
55640	2013-10-22 09:56:31	03950	22.539482	114.024010
55641	2013-10-22 09:44:26	03950	22.540451	114.119415
55642	2013-10-22 08:54:10	03950	22.542212	114.125252
55643	2013-10-22 09:57:15	03950	22.542219	114.125259
55644	2013-10-22 09:59:53	03950	22.542219	114.125264

Table 7 shows Smart Card dataset structure, where *Smartcad ID* represents the ID of the tapped card, Time represents the moment the tapping information was recorded, Transaction Type - in our case - indicates that is a bus boarding, and Bus Line has the line of the bus which has the Fare Collection System where the smart card data was recorded.

Table 7 - Smartcard dataset structure example

Smartcard ID	Time	Transaction Type	Bus Line
000000045	2013-10-22 08:18:55	31	M364
000000061	2013-10-22 08:17:53	31	385
000000070	2013-10-22 08:06:53	31	111路
000000131	2013-10-22 08:07:04	31	50路
000000140	2013-10-22 08:014:05	31	50路
000000076	2013-10-22 08:40:59	31	111路

Bus stops location dataset can improve the accuracy and permit us to obtain better results once we know exactly the bus stops where the passenger boarded. The structure is shown in *Table 8*, where Bus Stop ID represents the id of the station, Latitude and Longitude represent the bus stop position and Terminal indicates if the recorded bus stops is a bus terminal with connections.

Table 8 - Bus stops dataset structure example

Stop ID	Stop Code	Stop Name	Latitude	Longitude
1	TYVBT	Tao Yuan Village Bus Terminal	22.5325462	113.9200887
2	LHG	Long Hui Garden	22.567073	113.960272
3	CGV	Cha Guang Village	22.568309	113.943814
4	XWV	Xin Wu Village	22.5325462	113.9200887

5.1.3 Methodology

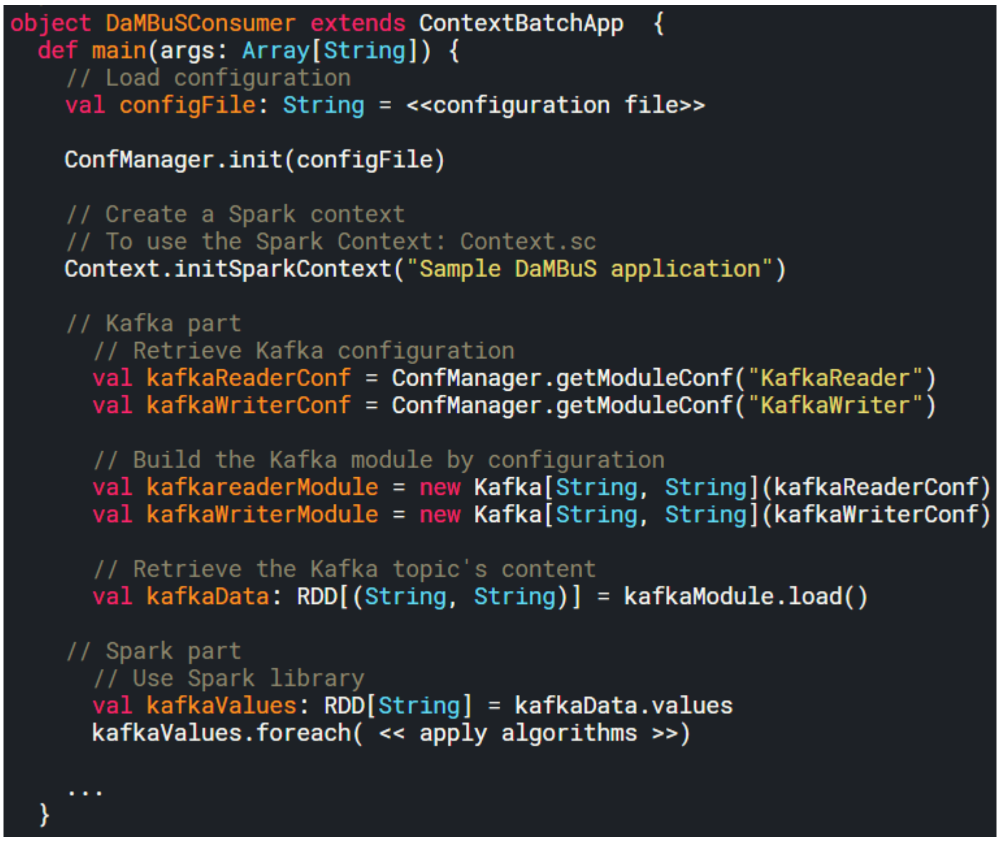
We can split our methodology in three main phases. The first one is the data pre-processing, where among the datasets we retrieve only the data we want to apply our experiments. After, in the second phase, we have the O-D matrix computation, responsible for finding out the origin and destination of passengers. At the end, in the third and last phase, is where stops and city zones are classified based on the correlation results. The next subsections describe all these phases, one-by-one.

5.1.3.1 Data pre-processing

The first part of this case of study consists in data pre-processing. As we want to put in place our architecture for smart transportation area, we will not process all the data, but we will only consider one specific line - B606. Therefore, all data not related to this line will be filtered out from the input data and not stored in Hadoop Distributed File System (HDFS). The choice of this line has not a specific reason, however after a deep analysis in the datasets it seems to be the one with more related data. Noises can be found inside datasets, and they are described and removed in section *4.3 Evaluation*. We could, though, have avoided the filtering part and stored all incoming data inside HDFS. This is normally what is done in Big Data approaches related to batch processing, where we first store everything and then we decide what is valuable for our problem.

5.1.3.2 O-D matrix

Having all valuable data stored in HDFS and available through Apache Impala, we can now start processing it for our use case. O-D matrix computation is the core of any density-flow problem and here it represents the boarding and landing bus stops of all passengers of line B606. First step is to ingest data into Apache Kafka, which is performed by a Kafka producer developed in Scala. As our current experiment uses batch mode, Kafka is not strictly necessary, however is part of our architecture and allows us to choose for a stream process in future work. From inside our batch processing component, *Algorithm 1* and *Algorithm 2* are executed through a Scala Spark job. This job reads from a Kafka topic where our data was ingested to and applies the algorithms. The result is our O-D matrix and allows us to compute passengers' density and flow. Above we show a pseudo-code of this job in *Figure 4*:



```scala
object DaMBuSConsumer extends ContextBatchApp  {
  def main(args: Array[String]) {
    // Load configuration
    val configFile: String = <<configuration file>>

    ConfManager.init(configFile)

    // Create a Spark context
    // To use the Spark Context: Context.sc
    Context.initSparkContext("Sample DaMBuS application")

    // Kafka part
      // Retrieve Kafka configuration
      val kafkaReaderConf = ConfManager.getModuleConf("KafkaReader")
      val kafkaWriterConf = ConfManager.getModuleConf("KafkaWriter")

      // Build the Kafka module by configuration
      val kafkareaderModule = new Kafka[String, String](kafkaReaderConf)
      val kafkaWriterModule = new Kafka[String, String](kafkaWriterConf)

      // Retrieve the Kafka topic's content
      val kafkaData: RDD[(String, String)] = kafkaModule.load()

    // Spark part
      // Use Spark library
      val kafkaValues: RDD[String] = kafkaData.values
      kafkaValues.foreach( << apply algorithms >>)

    ...
  }
```

Figure 6 - Scala consumer job responsible for consuming data from Kafka and applying the algorithms

5.1.4 Results

Technical results are obtained by applying the eight (8) different configurations described in section 4.1.1 – *Table 5*. By changing the configuration of input dataset size, number of Spark Workers and Spark Workers memory we can have a visibility about the scalability of our architecture. Workers are running Spark instances where executors live to execute tasks – compute nodes in Spark. In another hand, a Spark executor is a distributed agent responsible for executing tasks. The metrics used to compare the set-ups are in *Table 9*: execution time, stages, number of tasks and shuffle time.

Execution time represents the total time of the Spark job took. This time is spread among the job stages and tasks. Stage is a physical unit of execution, meaning a step in a physical execution plan. It is a set of parallel tasks – being one per partition – uniquely identified by an id. A Spark job can contain two different types of stages:

SuffleMapStage, intermediate stage that produces data for another stage; ResultStage, final stages that executes a Spark action by running a function on a RDD (Resilient Distributed Dataset). Figure 5 shows this split:

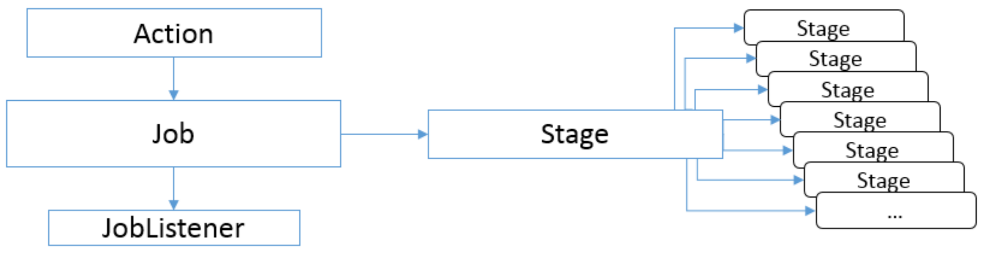


Figure 7 - SuffleMapStage and ResultStage inside a Spark job

Task represents the smallest unit of execution that is launched to compute a RDD partition and can be categorized into two different types: ShuffleMapTask, executes a task and divides the output to multiple buckets; ResultTask, that executes a task and sends the output to the driver application. Earlier stages and he last stage in a Spark job execution present ShuffleMapTasks and ResultTasks, respectively.

A Spark job runs stage by stage, where each stage is built up by DAGScheduler according to RDD's (Resilient Distributed Dataset) ShuffleDependency. Each ShuffleDependency maps to one stage in Spark job and then will lead to a shuffle. Shuffle is an expensive action as during it data no longer stay in memory. During a Spark execution, a shuffle process involves data partition, data compression and disk I/O. *Table 9* shows the compress dataset size and the average execution time of each one of the 8 different setups, and *Figure 6* shows an overview of a full Spark execution process. It is important to highlight that, as Spark loads all data in memory, it can be a limitation in terms of infrastructure.

Table 9 - Different setups performance comparison

Experiment	Local Size	HDFS Size	Workers/Cores	Execution time
1	400Mb	95Mb	1/8	~18s
2	400Mb	95Mb	3/8	~20s
3	4Gb	950Mb	1/8	~39s
4	4Gb	950Mb	3/8	~43s

5	20Gb	4.7Gb	1/1	~1.9min
6	20Gb	4.7Gb	1/8	~2.1min
7	20Gb	4.7Gb	3/12	~2.6min
8	100Gb	24.7Gb	3/12	~4.3min

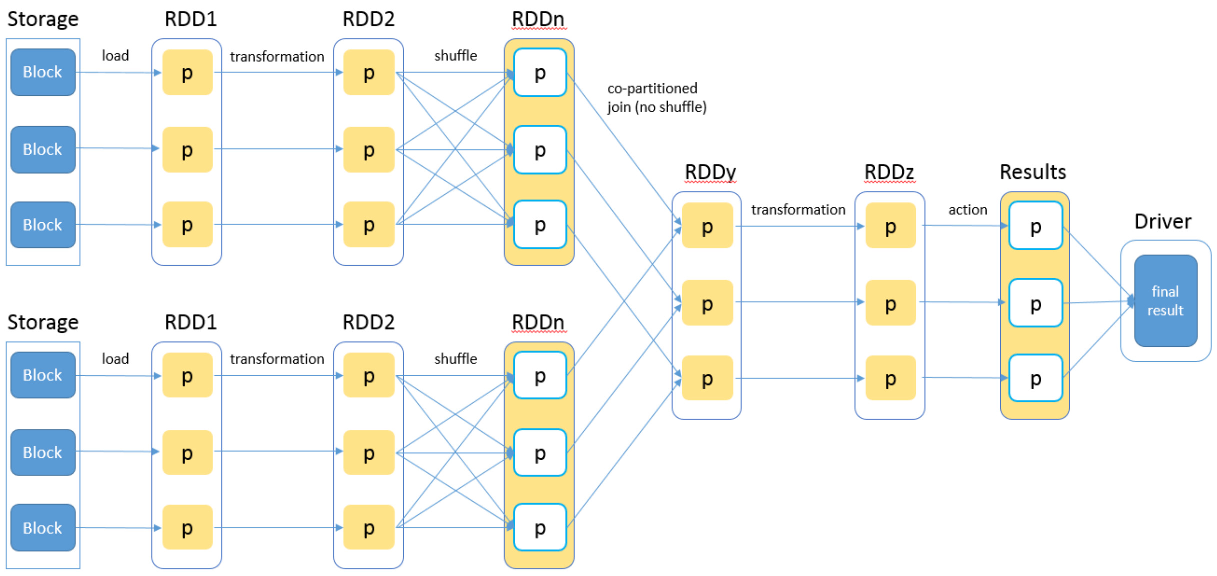


Figure 8 - Overview of a full Spark execution process

5.1.5 Final Results Evaluation

Spark jobs rely on memory, so the configuration of spark workers, executors, and workers memory mainly depend on the available memory during the execution. From results presented in 4.1.4 we can realize that experiment 8 was the most preformistic. As Spark loads all data in memory to avoid read/write operations, it also depends on the available memory. It may lead to an infrastructure limitation, if data cannot be loaded in memory. One solution is, whenever memory limit is reached, data is persisted in disk, which may decrease a bit performance – as I/O operations – however avoid data loss. We can see it also relies on the amount of input data Spark needs to process.

4.2 Data Analysis Experiments

In this chapter, we present our data analysis experiments. In a high-level overview, it means our Lambda Architecture implementation applied to the scope of bus management.

5.2.1 Experiments Set-up

From all set-ups described in section 4.1.1, here we consider only experiments 1 and 8. These experiments have the simplest and the most preformistic infrastructure, respectively.

5.2.2 Results

5.2.2.1 Origin-destination matrices

As described in section 3.3.1.1, O-D matrices are the start point for retrieving passengers density and flow. Having bus stops and smart cards latitudes and longitudes, we can correlate them – smart card *vs* bus stop – and identify in which bus stop a certain smart card – representing a passenger – has boarded in or boarded off. By applying the two algorithms described in *Figure 3* we had performed this correlation and obtained the tuples of smart card and bus stop.

In this work, we also analyzed, among the origin-destination matrix, how many of the total tuples represent a simple trip, connection trip and round-trip – section *4.2.3.* In total, we identified 3988 tuples.

5.2.2.2 Density and Stops Classification

Having O-D matrix allows us to compute the passengers density and flow, moreover to start the classification process among bus stops and city zones. Classification is done based on four pre-defined categories: *residential, work, nightlife, personal.*

First step consists in classifying all GPS records in their respective stops. Each GPS record coordinates are compared to all bus stops coordinates. It was done using another Scala Spark job running inside Hadoop, in Lambda Architecture. For the experiment, the day intervals were divided in: *residential (*6am-8am), *work* (5pm-8pm), *nightlife* (11pm-5am) and *personal. Personal* category encompasses all the remaining periods not classified in the others. The target bus stop is the one that presents the minimum distance difference. Then, we had the passenger density computation and bus stop main travel purpose discovery.

Our implemented Lambda Architecture, although able to handle Big Data use cases, was applied to a considerable big amount of data – but not as huge as in a Big Data use case. However, using it and using implemented Spark jobs, all smart cards had been linked to a bus stop and also related to a travel purpose. This classification helps to understand what are the travel purposes associated to each stop and, based on the stop geolocation, what are purposes of city zones. This is not the main goal of this work, however having O-D matrix results allows us to do so, improving the understanding of the bus network demand

5.2.3 Final Results Evaluation

Evaluation can be split in technical evaluation - where we present the performance of using Lambda Architecture implementation, specially Spark and Hadoop comparing to standard approach - and the scenario evaluation - where we present the result of our architecture applied passengers density computation and classification.

Taking the technical evaluation, we can a priori infer what Big Data state of art mentions – section 3.1. Spark is a framework that comes after the first Hadoop approach - MapReduce - reducing hard disk usage and increasing performance in around 100 times and 10 times faster than the MapReduce in disk and in memory, respectively. Unfortunately, our baseline - and any other in literature - goes into details related to performance and processing time. However using all scalability and parallelism of our architecture, we show here some performance metrics as Spark jobs execution time and Hadoop size. The technical evaluation is the most important one in this work, as our main goal was to implement a framework based on the Lambda Architecture approach, using the top and low-cost Big Data technologies available currently.

The experiment evaluation results - focusing on the problem approach – are described in the three graphics below. First, for O-D matrix computation we find out boarding and landing stops of each passenger - each record inside smart card dataset - and afterwards the passenger density.

Figure 7 demonstrates how the cards are distributed among the travel purposes. From total 3186 cards registers analyzed, 1350 – 42.4% - represent *residential* as the main travel purpose. *Work* appears as the second main reason for taking a bus, with 1020 – 32% - departing buses, and *personal* appears with 816 – 25.6% - records. It is important to mention that the *personal* category was added in this first experiment as the datasets represent only one day in the bus service network. *Night* purpose did not have any record. As the used dataset is of one day in Schenzhen, and it was a business day - Tuesday - that may be the reason for we do not find any passenger taking a bus for a *nightlife* purpose.

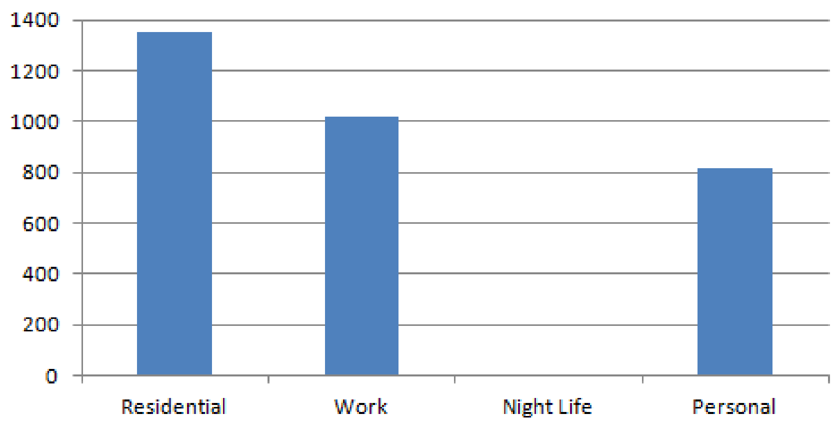


Figure 9 - Smart cards categorization

Figure 8 shows the passenger density in each bus stop. *LHD* is the densest stop, with 1551 records, being 379 of *work* type and 274 of *residential* type. *KFR* appears as the less dense stop, with only 97 passengers leaving the stop, 22 of them with *work* as travel purpose and 24 with *residential* as travel purpose. *Figure 9* shows the density of the second graph distributed between three travel purposes: *work*, *residential* and *nightlife*. Again, we see *LHG* stop with the highest amount. From that graph, we can notice that this stop is the one that has more passengers leaving the station in the peak periods – 6am to 8am for *residential* and 5pm to 7pm for *work*, which result in the highest passenger density shown in the second graph.

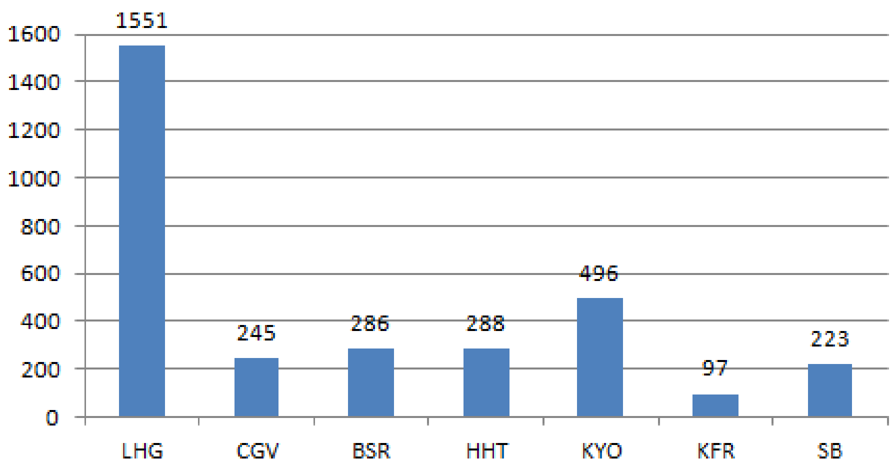


Figure 10 - Passengers density among bus stops

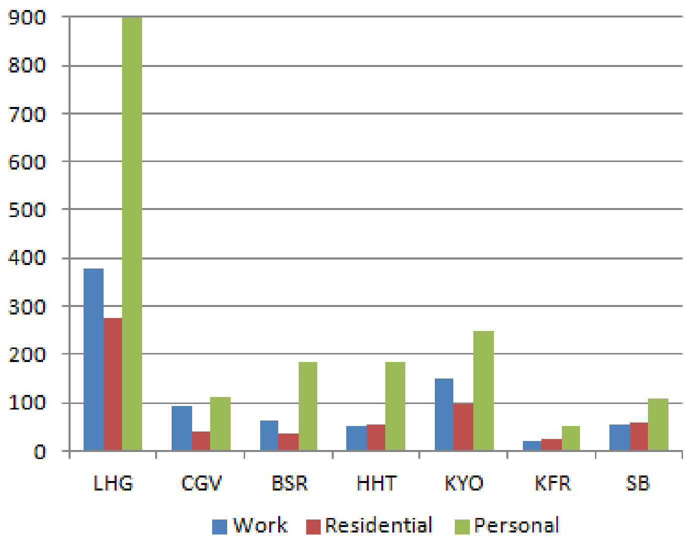


Figure 11 - Passengers density per travel purpose per bus stop

Differently from (ZHANG, 2014), this work also considers the scenario where passengers tap the card more than once. Taking into account the intervals shown on Table 6 for trip type definition based on number of card taps, the origin-destination pairs in the dataset were also classified between the three types: *simple trip, connection trip, round-trip.* All cards registers were divided into these three types, totalizing 2134 *simple trips,* 1604 *connection trips* and 250 *round-trip - Figure 10.* We have to mention that the maximum number of connections found in a trip was 2. Therefore, 1064 connection trips represent 802 final destinations. At the end, that represents 3186 trips. We This result is an important way to understand how passengers move in their trips, as sometimes they may get more than one bus to reach their destination, or the purpose of the trip is a round-trip.

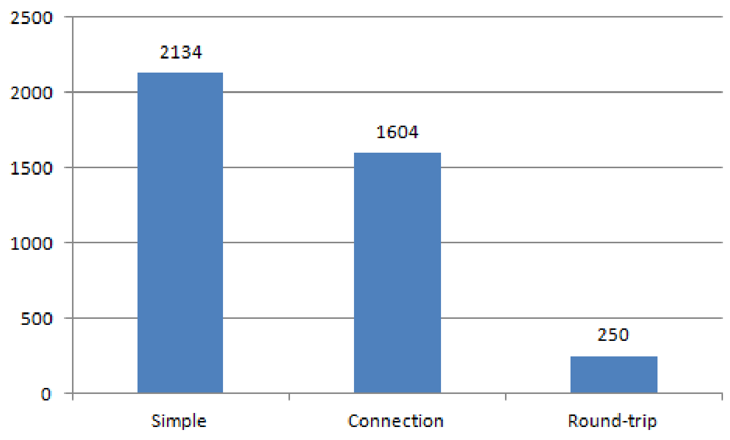


Figure 10 – Travel type

Our main objective during this work is to expose the idea of the Lambda Architecture, presenting its potential and giving a path for implementing it. We applied our architecture in bus service management area. We can highlight that our case of study does not represent a real big data scenario, as our dataset is limited to 100Gbaims to cover an end-to-end scenario, using all technologies available in our architecture. This limitation was also in terms of hardware and memory.

6 CONCLUSION

In this work we presented LDAVI, a Lambda Architecture Driven Implementation based on Lambda Architecture approach (KIRAN, 2015), a data-processing architecture for handling massive amount of data by decomposing the problem into three layers: batch layer – for historical data processing - serving layer and speed layer – for streaming processing.

Our objective was to implement a low-cost framework based on Lambda Architecture and demonstrate it applicability by applying it in a real scenario in Smart Mobility area, extracting passengers density and flow through Smart Card, bus stop geolocation and buses GPS data.

The improvement on public transport quality influences directly and positively in society. Furthermore, it brings countless benefits for people lives as well as helps to solve urban mobility problems that are current present in big cities. Knowledge about public transport network behavior allows decision-making, increasing, service quality, passenger experience, usage and profits. We used our architecture to solve a Smart Mobility challenge – passengers' density and flow, using concepts of Big Data, Smart Cities and Lambda Box Architecture – Apache Kafka, Hadoop and Spark - the model computes passengers' density and flows using time-matching methods and clustering. Differently from related works, we approached three different types of trip: simple trip, connection trip and round trip, what makes the analysis complete and more accurate. As used datasets were data-limited, the experiment did not represent the whole scope of this project, but a part of it. However, it demonstrates the feasibility of what is proposed; showing that understanding passengers' behavior, travel proposals, density and flow are an important source of knowledge to improve public bus service.

We have already presented our contribution "Passenger density and flow analysis and city zones and bus stops classification for public bus service management" in the Brazilian Symposium on Databases (SBBD) in 2016 and "Data Mining Framework for Bus Service Management: Passenger's density and flow analysis, city zones and bus stops classification" in the International Conference on Enterprise Information Systems (ICEIS) in 2017 as short paper. In these contributions, the focus was in the application of our framework and not the framework itself. We have realized that the usability and powerfulness of the framework presented in this work could be used not only in Smart Mobility, but in any problem requiring a Big Data approach.

A short-term improvement to this work will be to better cluster the days by considering business days, weekends and holidays. This will help the decision-making as travels purposes may change if done in a business day or weekend and eventually *Nightlife* category may be considered only for weekends. Future work will apply this architecture to a bigger Smart Mobility challenge in terms of amount of data. Moreover, we plan to implement a full Lambda Architecture, containing its batch layer – as built in this work – for processing huge and historical amount of data of public transportation and a speed layer, for processing in streaming mode. The idea is to build the speed layer by using Apache Spark Streaming and to apply it in public transportation problems requiring real-time computation and analysis (such as bus scheduling and passenger tracking).

6 REFERENCES

UNITED NATIONS. **Population Distribution, urbanization, internal migration and development: An international perspective.** 2011.

IPEA - INSTITUTO DE PESQUISA ECONÔMICA APLICADA. Disponível em: < http://economia.uol.com.br/ultimas-noticias/efe/2011/05/04/ipea-diz-que-65-dos-brasileiros-das-grandes-cidades-usam-transporte-publico.jhtm>. Acesso em: 15 out. 2016

PELLICER, S. et al. **A Global Perspective of Smart Cities: A Survey.** 2013.

MARZ, Nathan et al. **Big Data: Principles and best practices of scalable realtime data systems. Manning Publications, 1 edition**. 2013

NASIBOGLU, E. et al. **Origin-Destination Matrix Generation Using Smart Card Data: Case Study for Izmir.** 2012.

QING, Z. et al. **Public Transport IC Card Data Analysis and Operation Strategy Research Based on Data Mining Technology.** 2009.

LI, Man et al. **Public Transport Smart Card Data Analysis and Passenger Flow Distribution.** 2012.

DUAN, W. et al. **Analysis of Single-line Passenger Flow Based on IC Data and GPS Data.** 2012.

ZHANG, J. et al. **Analysing Passenger Density for Public Bus: Inference of Crowdedness and Evaluation of Scheduling Choices.** 2014.

KIEU, Le Minh; Bhaskar, Ashish; Chung, Edward. **Passenger Segmentation Using Smart card Data**. 2015.

GUIDO, Giuseppe et al. **Big data for public transportation: A DSS framework.** 2017

KIRAN, Mariam et al. **Lambda architecture for cost-effective batch and speed big data processing.** 2015.

VILLARI, Massimo et al. **AllJoyn Lambda: An architecture for the management of smart environments in IoT.** 2014

GRIBAUDO, Marco et al. **A performance modeling framework for lambda architecture based application.** 2018

MUNSHI, Amr et al. **Data Lake Lambda Architecture for Smart Grids Big Data Analytics.** 2018

YANG, Fangjin et al. **The RADStack: Open Source Lambda Architecture for Interactive Analytics.** 2017

BRIAND, Anne-Sarah et al. **Analyzing year-to-year changes in public transport passenger behaviour using smart card data.** 2017

ERATH, Alexander et al. **Transport modelling in the age of big data.** 2017

GU, L. et al. **Memory or Time: Performance Evaluation for Iterative Operation on Hadoop and Spark.** 2013.

ZAHARIA, M. et al. Fast **and Interactive Analytics over Hadoop Data with Spark.** 2012.

LI-JUN, Q. et al. **Evaluation of reliability of bus service based on gps and smart card data.** 2011.

SUN, L. et al. **Using smart card data to extract passenger's spatio-temporal density and train's trajectory of mrt system.** 2012.

ZHENLIANG, M. et al. **Predicting short-term bus passenger demand using a pattern hybrid approach.** 2014.

SCHENZHEN, CHINA SAMPLE DATA DESCRIPTION OF mPAT. Disponível em: <http://cloud.siat.ac.cn/mpat/>. Acessado em: 10 mar. 2016.

Big Data: Hadoop, Business Analytics and Beyond. Retrieved April 16, 2014, from http://wikibon.org/wiki/v/Big_Data:_Hadoop,_Business_Analytics_and_Beyond