

# **Big Data analytics infrastructure for easier urban planning**

*Vicky Dineshchandra*

A dissertation submitted in partial fulfillment  
of the requirements for the degree of  
**Masters of Engineering**  
of  
**University College London.**

Department of Computer Science  
University College London

April 29, 2018

I, Vicky Dineshchandra, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the work.

# Abstract

This dissertation investigates a "Big Data" infrastructure that provides investigative and analytical tools, in the context of urban planning where datasets are growing rapidly whilst urban planners struggle to keep up with new and emerging technologies. The project is supported by the Urban Dynamics Lab (UDL) who are based in University College London (UCL).

This project experiments with using Jupyter Notebook as an abstraction layer on top of a Spark-Hadoop computing cluster to enable urban planners to allow access their datasets in a more familiar and simplified manner. This is in contrast to traditional practises where a comprehensive understanding of distributed systems is required to leverage the computational benefits of "Big Data" technologies.

To conduct this experiment, a real world, user facing three node "Big Data" computing cluster is set up (as would be used by the UDL). The Jupyter Notebook-Spark interface then is installed. In order to test the interface, an urban planning question is proposed and then answered by utilising the Jupyter Notebook-Spark interface and several data transformation pipelines which have also been implemented. This simulates UDL researcher's workflows and acts as a proof-of-concept. Results of this study show that the Jupyter Notebook-Spark interfaces allows urban planners to write analytics program faster and more freely compared to conventional programming frameworks such as MapReduce or Spark.

This research investigates a potential interface to better enable "Big Data" access to researchers and urban planners that may not have verbose infrastructure knowledge. Furthermore, the outcomes of this project will be directly used to facilitate UDL's research.

# Acknowledgements

This dissertation would not be possible without the help and guidance of Prof. Philip Treleaven. I would also like to thank my technical supervisors Dr Michal Galas and Dr Engin Zeynep for their expertise and enthusiasm throughout this project.

# Contents

<b>1</b>	<b>Introduction</b>	<b>9</b>
1.1	Research Motivation . . . . .	9
1.2	Research Objective . . . . .	10
1.3	Structure of this dissertation . . . . .	11
<b>2</b>	<b>Background &amp; Literature Review</b>	<b>12</b>
2.1	Definition Of Big Data . . . . .	12
2.2	Appropriating a distributed file system . . . . .	13
2.3	Computation on a distributed scale . . . . .	14
2.4	Spark . . . . .	15
2.5	Distributed Databases . . . . .	16
2.6	Handling Real Time events . . . . .	17
2.7	Scheduling . . . . .	18
2.8	Industry Applications . . . . .	19
<b>3</b>	<b>Project Requirements</b>	<b>20</b>
3.1	The Urban Dynamics Lab . . . . .	20
3.2	Platform Overview . . . . .	20
3.3	Jupyter Notebook web interface . . . . .	21
3.4	A practical use case . . . . .	22
3.5	ETL Pipelines . . . . .	22
3.6	Formal specification . . . . .	23
3.7	Software Methodology . . . . .	23
3.8	Version Control . . . . .	24
<b>4</b>	<b>Urban Planning Datasets</b>	<b>25</b>
4.1	London Development Database . . . . .	25
4.2	London Air Pollution API . . . . .	27
<b>5</b>	<b>System Implementation</b>	<b>28</b>
5.1	Fixing the UDL Cluster . . . . .	29
5.1.1	Outline of the UDL cluster . . . . .	29
5.1.2	Problem . . . . .	29
5.1.3	Solution . . . . .	29
5.2	The London Development Database Pipeline . . . . .	29
5.2.1	Existing problems . . . . .	29

5.2.2	Solution: Docker . . . . .	30
5.3	The London Air Pollution Pipeline . . . . .	31
5.3.1	Existing problems . . . . .	31
5.3.2	Suitable programming languages . . . . .	32
5.3.3	Pipeline walk through . . . . .	32
5.4	Automated Pipelines with Oozie . . . . .	33
5.5	Jupyter Notebook with Spark Integration . . . . .	34
5.6	Bringing it all together: A practical case-study . . . . .	35
<b>6</b>	<b>Testing &amp; Results</b>	<b>37</b>
6.1	Answering an urban planning question . . . . .	37
6.2	The Jupyter Notebook Service . . . . .	39
6.2.1	Build Testing with Docker . . . . .	39
6.2.2	Results . . . . .	39
6.3	The London Development Database Pipeline . . . . .	39
6.3.1	Unit Testing with Nose . . . . .	40
6.3.2	Results . . . . .	40
6.4	The London Air Pollution Pipeline . . . . .	40
6.4.1	Unit Testing with JUnit . . . . .	40
6.4.2	Build Testing in Docker . . . . .	40
6.4.3	Results . . . . .	41
<b>7</b>	<b>Conclusion &amp; Future Works</b>	<b>42</b>
7.1	Summary . . . . .	42
7.2	Evaluation . . . . .	43
7.3	Future Work . . . . .	44
	<b>Appendices</b>	<b>45</b>
<b>A</b>	<b>Source Code</b>	<b>45</b>
<b>B</b>	<b>Project Plan</b>	<b>46</b>
<b>C</b>	<b>Interim report</b>	<b>48</b>
<b>D</b>	<b>UML Diagram for AirPollution API</b>	<b>50</b>
	<b>Bibliography</b>	<b>51</b>

# List of Figures

2.1	A simple example of a MapReduce algorithm . . . . .	14
2.2	A logistic regression benchmark test with Spark and Hadoop . . . . .	15
2.3	Overview of how Spark Streaming handles incoming streams. (Note that a node can accept several RDDs and is not a 1:1 correspondence.) . . . . .	18
3.1	Overview of UDL's entire data analytics platform . . . . .	21
3.2	The timeline of the project as a Gantt Chart. . . . .	24
4.1	Public web interface for LDD . . . . .	25
4.2	The many tables and views that exist in the LDD dump, most without documentation. . . . .	26
5.1	Node structure of the UDL cluster . . . . .	28
5.2	All running services on the cluster . . . . .	30
5.3	DAG for the Air Pollution pipeline . . . . .	33
5.4	DAG for the London Development pipeline . . . . .	34
5.5	Final visual representation of construction site data against air pollution . . . . .	36
6.1	Final map of overlaid dataests . . . . .	38

# List of Tables

- 2.1 An example HBase table. Note that only the Column Families are fixed, the columns in the Column Families can dynamically altered. 17



## **Chapter 1**

# **Introduction**

This chapter presents an overview of the dissertation. First, the motivation behind this research is discussed to explore the problem area and provide context to the reader. Based on the issues discussed, an objective is formalised to make clear the purposes of this project. Finally, the chapter ends by outlining the different chapters that make up the rest of this dissertation.

### **1.1 Research Motivation**

There is an increasing demand for services that can deal with growing amounts of data. In just the last two years, more data has been produced than in the entire history of the human race [1]. It is predicted that there will be 44 times as much data produced in 2020 compared to 2009 [2]. Research into developed European economies show that the inability to leverage Big Data costs governments 100 billion Euro [3].

Researchers need to be well equipped and have access to the right tools to be able to keep up with the Big Data trend. In this project, we focus on the Urban Dynamics Lab (UDL) who are developing and innovating new tools for their researchers and in particular, urban planners, have better access to Big Data analytics tools.

The Urban Dynamics Lab was set up at University College London (UCL) and aims to bring together expertise from a wide variety of sectors such as the UCL Geography department, Centre for Advanced Spatial Analysis (CASA) and the UCL Computer Science department [4]. It is funded primarily by the UK Regions Digital Research Facility.

The goal of UDL is to make better urban planning decisions through data analytics and urban modelling. One of the ways in which UDL aims to make this possible is by creating a large data analytics platform accessible to the members of

UDL. Usually, data analytics platform that can deal with Big Data utilises a variant of Apache Hadoop, Spark services. Unfortunately, the use of these services require a deeper understanding of distributed systems and verbose technical knowledge which most urban planners do not possess.

The solution to processing enormously large sets of data has existed for a number of years. Namely, Hadoop MapReduce, a widely-used open source framework has been around for years. This was inspired from the very first Google Distributed File System (GFGS). Hadoop usually refers to it's distributed file system, the Hadoop File System, which distributes the data across physical servers, whilst MapReduce jobs are written to perform fast operations. However, MapReduce jobs written in a way conforming to the Map Reduce paradigm which adds a barrier of entry.

In addition, the more recent Spark framework provides a more general compute interface and is considerably faster than Hadoop MapReduce. However in both cases, specific applications have to be written to use the Big Data technologies. This usually involves having knowledge of distributed systems, the Java programming language, the Scala programming language, a flavour of Linux and functional programming paradigms.

These requirements can be overwhelming for an urban planner and be too much of an hurdle to be able to analyse the dataset. This prompts us to provide better interfaces for urban planners, such as a Jupiter Notebook interface, a Python programming environment which most analysts are familiar with.

## 1.2 Research Objective

The objective is to improve the accessibility of Big Data analytic tools for urban planners by creating an interface that urban planners are already comfortable with. The main hypothesis states that:

- By using an a Jupyter Notebook-Spark interface, urban planners will be able to write analytics code in a familiar environment (namely in a Jupyter Notebook setting with Python code) whilst leveraging the computational benefits of a distributed computing framework such as Spark.
- To validate this hypothesis, we will conduct an experiment where there exists a Jupyter Notebook layer on top of a Hadoop-Spark cluster. A urban planning "question" will be answered using the Jupyter Notebook interface to show the capabilities of the Jupyter Notebook interface.

## 1.3 Structure of this dissertation

This dissertation is structured as follows:

- **Chapter 2** looks at existing literature around Big Data technologies as well as some of the popular software packages
- **Chapter 3** gives more detail about the stakeholders and the formal requirements of the project.
- **Chapter 4** describes the datasets being used in the experiment and how they are appropriate in the context of urban planning data.
- **Chapter 5** describes the finer implementation details of the experiments and subsequently, dwells into more depth around the system architecture and software specifics used to build the system.
- **Chapter 6** describes the methodology used to test the effectiveness of the experiment and outlines the results discovered.
- **Chapter 7** summarises the findings of this dissertation and provides potential possibilities for future work.

## Chapter 2

# Background & Literature Review

This chapter explores the mechanics of Big Data technologies. In particular, this chapter presents an overview of design principles and architectures that allow us to compute on larger sets of data compared to traditional methods. Finally, the chapter finishes by briefly looking at some industry specific applications and current usage.

### 2.1 Definition Of Big Data

Let us begin by defining the term "Big Data" which has become somewhat of a buzzword in the media [5]. Traditionally, we usually have some collection of data stored in a single database on a single server. This sole server is responsible for querying some set of data, manipulating it and retrieving information. This is a common practice that has been used successfully across academia and industry and is still a good strategy provided that the size of the data is small, i.e not in the tens of terabytes [6].

However, as data continues to grow at an exponential rate, the size of the dataset begins to enter the terabyte range (1000 gigabytes). Whilst one could still store all this information in a single server, querying this would be computationally slow and incredibly expensive (both in terms of time and hardware costs). When a dataset becomes large enough that it cannot be handled by a single machine, it can now be deemed "Big Data"[7].

We should also consider other characteristics of a dataset other than it's size. These are often referred to as the "Five Vs" [8]:

- **Volume** - the quantity of stored data.
- **Variety** - the type and nature of the data
- **Velocity** - the speed at which data is generated

- **Variability** - the inconsistency of the dataset
- **Veracity** - the quality of the dataset

Given such characteristics, the challenge now becomes:

- How does one build a system capable of handling a dataset as described by the 5 Vs.

## 2.2 Appropriating a distributed file system

Whilst the term "Big Data" has been around since 1990 and there have been many attempts at building a distributed solution to the problem. However, in 2003, Google released a white paper on the Google File System (GFS), a key moment in Big Data history. The paper provides insights into some of the design decisions for the GFS which subsequently allow it to serve Google's large-scale data processing requirements [9]. The GFS whitepaper would go on to become the basis for the widely used, open source framework Apache Hadoop. As this project (and many other industry grade projects) makes use of Hadoop, let us assess some of the key features of the GFS.

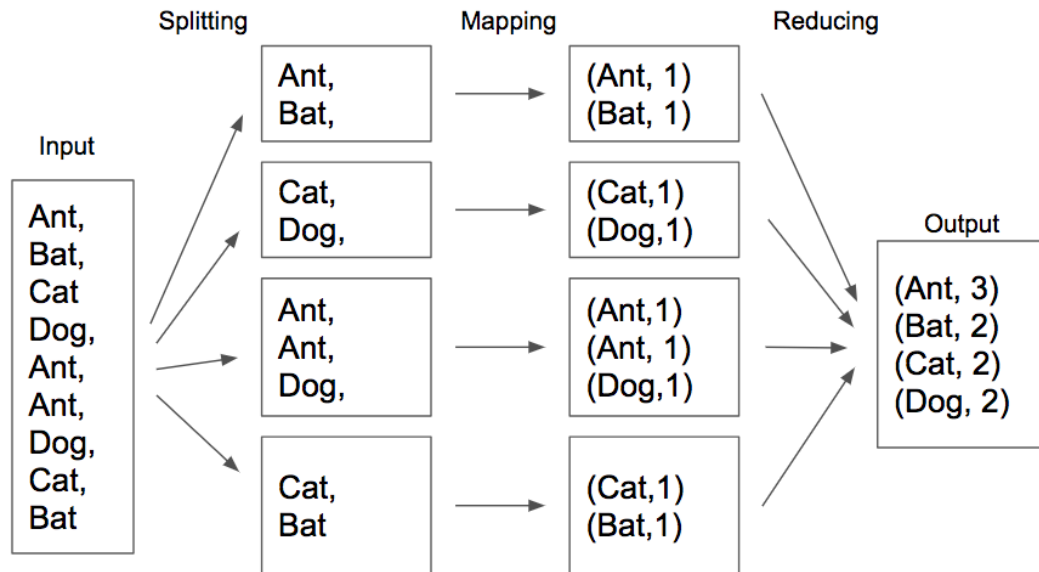
One key feature of the GFS is that it is built from cheap, inexpensive commodity hardware [9]. Commodity hardware is more readily available to companies than state of the art, expensive, powerful machines. However, commodity hardware is also more likely to fail so the system must be able to detect faults and replicate data for redundancy. This is a useful requirement in the context of this project also.

In addition, the GFS is designed such that each server holds a modest amount of larger files ranging from 100Mb to a few GB[9]. Certainly with this project, most data files will be quite large, rather than having a numerous amount of small files. As the GFS is optimised for large files, it's use does work in our favour.

The GFS cluster has a single *master* server connected to several *chunkserver*s servers. The role of chunkservers is to store replicate data on their local disks and provide extra reliability. The default configuration consists of three chunkservers to a master server although this can be set by the developer [9].

The role of the master is to direct client connections to the server with the information required to deal with the request. To do this, the master server holds all metadata about the cluster set up. More importantly, the master server never retrieves the data for the client connection, as this would become a bottleneck [9].

The GFS has been designed to account for constant server failures, including the master server which is why data is replicated three times[9]. The master state is



**Figure 2.1:** A simple example of a MapReduce algorithm

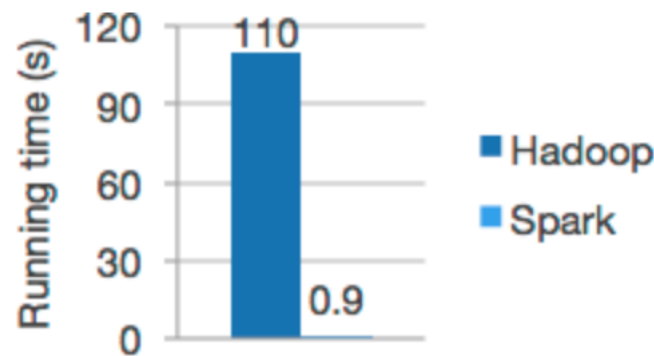
also replicated and in the event the master server fails, a chunk server takes on the responsibility of being the master, causing the system to recover seamlessly.

Many of the design requirements for the GFS are similar to other Big Data use cases and so the Apache Hadoop framework was created, with the GFS whitepaper as the inspiration. The Hadoop project started in 2006 and was named after one of the founder's son's toy elephant. This project is freely available under an open-source under the Apache software foundation [10].

## 2.3 Computation on a distributed scale

Apache Hadoop consists of several key components: the Hadoop Distributed File System (HDFS), Hadoop Yarn and Hadoop MapReduce. The HDFS has similar features to the GFS and allows for distributed storing of large amounts of data. Hadoop Yarn is a resource manager that works to schedule tasks. One of the greatest strength's of Hadoop is it's MapReduce functionality which allows for large-scale computation on distributed systems.

MapReduce in essence, is a programming model, first introduced by Google [11]. The MapReduce programming paradigm works as follows. Given a large dataset, we aim to perform as many independent operations (e.g filtering) as possible in a parallel fashion. This is known as the "Map" step [12]. The operations are stateless, meaning that they can be done very quickly as they have no dependencies on any other operation. Once the transformations have been applied, the data is then "Reduced" by a summarising operation (e.g counting) [12].



**Figure 2.2:** A logistic regression benchmark test with Spark and Hadoop

For example, in Figure 2.1, we show a counting example on a simple list. First, the list is split and sent to multiple nodes in the cluster to compute on. The list is first mapped, assigning a count of one to the score. Then, it's reduced by combining all the mappings from the nodes to give a combined result. For very large sets of data, it's easy to see how the splitting and mapping procedure provides a speed boost, and on reduction, it's quite fast as most of the calculations have already been done [13].

It's important to note that the MapReduce paradigm provides a computational advantage under certain conditions. Firstly, the operations must be parallel and so at least it should be compatible with multi-threading. This notion, when extended to massive distributed systems, it's easy to see how parallel operations happening on different machines can provide a speed gain. Secondly, the Map-reduce programming paradigm requires a good algorithm that conforms to the Map-then-Reduce model. It makes the most amount of sense to do all the Maps - the independent, stateless operations - at the beginning and the state-full operations follow [13].

One of the limitations of this is that we have to configure our operations to be in the MapReduce paradigm which is not always a natural way of computing or expressing computation. This can mean that time is spent polishing MapReduce algorithms for efficient.

## 2.4 Spark

Spark is an open-source general computing framework that was started in UC Berkeley and then adopted into the Apache foundation [14]. Whilst Spark can be used with Hadoop, Apache Spark outperforms considerably, as shown in Figure in a logistic regression cluster benchmark [15].

Spark outperforms Hadoop because it keeps most of the data in (RAM) memory, as a pose to disk like Hadoop does. To do this, it uses a in-memory data structure called a Resilient Distributed Dataset (RDD). This makes retrieving data, transforming it and processing it far faster than Hadoop. Data can also be cached in Spark but not in Hadoop which gives it a massive speed boost. It still follows a distributed paradigm in that the RDDs are networked and synchronised between all the nodes in the cluster.

Moreover, Spark is also more versatile than Hadoop MapReduce in terms of the jobs it can provide. MapReduce jobs are mostly batch jobs, they run for a predefined period of time. However, Spark can deal with real time streams, database operations, providing a more general compute framework [14].

One of the key advantages of Spark is that the developer no longer needs to adapt their algorithms that follow the MapReduce paradigm. Another major improvement this provides a massive speed advantage compared to traditional Hadoop MapReduce functionality. Spark can be used to do computations on top of Hadoop and mixed with other Big Data solutions too.

## 2.5 Distributed Databases

Whilst HDFS provides a distributed storage solution and a general file system, there are also more purpose built distributed databases.

When designing a distributed database, there are two key factors that are in contention: replication and duplication. Duplication is a simple process, the database is duplicated across the nodes in a cluster at set interval to ensure protection against data being lost. Replication, however, requires software which looks at new changes or updates in the database and looks to replicate them across the nodes in the cluster. This can be quite challenging when multiple changes and need to be resolved during the replication process [16].

Whilst distributed databases enjoy similar benefits as distributed file systems, there are some unique issues that arise from it. With traditional databases, only a single database needs to be kept secure, here the database is scattered and fragmented, so we need to secure every single fragment which becomes a difficult task [17]. Naturally, this also invokes a lot of complexity in the design and usage of these systems compared to traditional databases. Furthermore, the cost of running so many machines and maintaining them can get rather large which is why larger companies are able to afford this.

Based on the Google Big Table paper, HBase is one example of a distributed database [18]. HBase can be thought of as a multi-dimensional hash map [19]. It



RowEntry	Column Family A			Column Family B	
	Column A	Column B	..	Column A	..
row	null	(key:value)	..	null	..
..	..	..	..	..	..

**Table 2.1:** An example HBase table. Note that only the Column Families are fixed, the columns in the Column Families can dynamically altered.

stores key-values pair, so it's similar to NoSQL databases. It has some unique features too. Instead of providing tables with columns and rows, it provides column families, which can hold many columns, that can be dynamically created and manipulated as long as the parent column family exists as shown in Table 2.1. Column families are the way that HBase ensures fault tolerance by using them as a basis for replication in case the case of system failure [19].

Due the fast look up speeds and slower write speeds, HBase is often used to store data that needs to be quickly accessed and less-frequently modified [20]. It also provides a versioning system which is part of it's fault tolerance system. With versioning, it stores previous records of the same entries upto a certain number of days. This means that the history can be linearly searched if needs be and provides more options to the developer [20]. Due to the unusual properties of HBase, the situation it is used in becomes important to fully leverage it's capabilities.

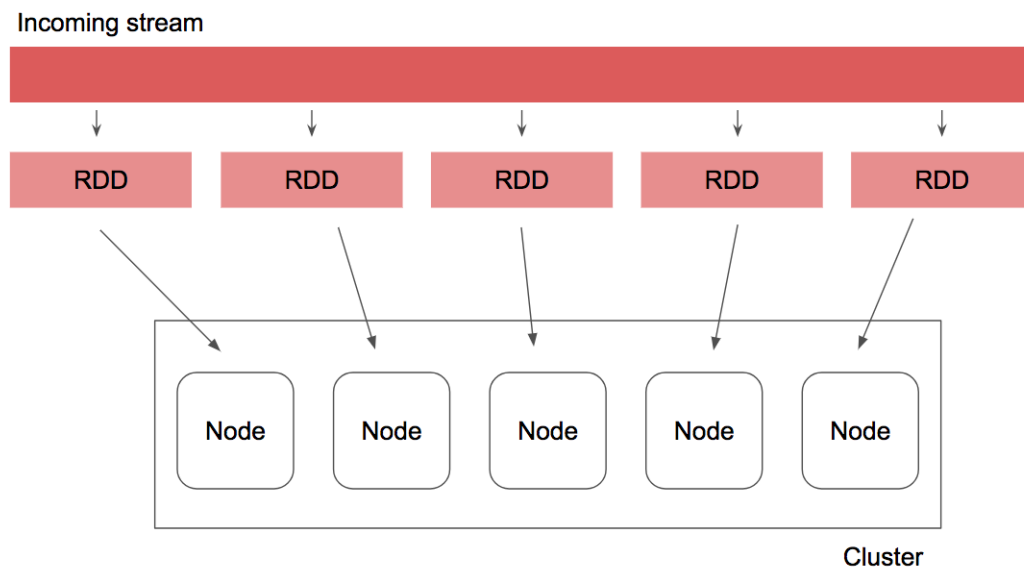
Apache Cassandra is also a distributed No-SQL database and provides high fault tolerance [21]. One of the biggest differences between HBase and Cassandra however, is that the latter requires structured data. Whilst this is a drawback in situations where the data is volatile, the rigour that is enforced brings about huge performance gains [22]. When scaled, Cassandra outperforms HBase on performance, although writing entries is still relatively slow [23].

There are also other distributed databases that are emerging, however all the current options are NoSQL based. Unfortunately, "end" users (such as analysts) and urban planners tend to be more comfortable with writing SQL queries [23]. As a result, there are tools that abstract these NoSQL databases and enable to execute SQL queries in a distributed fashion. Examples include Apache Impala[24] and Apache Phoenix[25].

## 2.6 Handling Real Time events

There is also the added complexity of data that is generated in real-time that systems need to cope with.

One of the most popular ways to deal with this is the Apache Spark Streaming



**Figure 2.3:** Overview of how Spark Streaming handles incoming streams. (Note that a node can accept several RDDs and is not a 1:1 correspondence.)

framework [26]. The framework itself can accept a socket connection and listen on a stream. Spark Streaming breaks down the streams into an abstracted data structure called a Resilient Distributed Dataset (RDD) [14] which then processed in parallel across the cluster. This allows large, dense streams to be processed in a distributed manner. An example of this is shown in Figure 2.3.

One limitation is that one has to operate on an RDD state-lessly, i.e there is no connection between that RDD operation and any other RDD operation. This can make working with streams difficult if the developer wants to perform state-full operations across the entire stream.

Apache Kafka is a solution built entirely to be able to deal with streams [27]. It uses a publisher/subscriber model where nodes can be publishers (output some data) or a subscriber (accept some data) [28]. Kafka streams are usually set up so they conform to some topic. Kafka nodes can then subscribe to a topic to receive all the information regarding a specific topic [28]. This can help sort streaming data and provide structure on a real time stream .

## 2.7 Scheduling

Manually configuring, running and waiting for MapReduce or other jobs can be a lengthy process. Due to the size of the datasets, it can take more than 10 hours to complete a job, which falls outside normal industry working hours. Big Data applications, as we have seen, have multiple dependencies which become difficult

to manage. Schedulers are used to pipeline jobs and handle dependencies.

To particular interest to the stakeholders in this project (explained in the next chapter) is Apache Oozie[29]. Oozie is a task scheduler that with tasks organised through a direct acyclic graph (DAG). Oozie has native support for all Hadoop based MapReduce, Streaming, Pig or Hive jobs as well as the ability to run Java programs and Shell scripts.

By modelling the executable actions as a DAG, it utilises the property of DAGs that there are no cycles [30]. This means that at no point in the workflow can the system find itself in a loop. This is incredible useful in Big Data systems to ensure that resources in the system are kept free as much as possible to allow for optimal throughput.

Tasks can be easily defined through XML files and modelled as a DAG to run. If tasks fails, there are dependencies and cases which can be defined in response to the dependency failure or runtime error.

## 2.8 Industry Applications

With the exponential increase in the amounts of data, Big Data applications have had an impact on all major industries. Due to the large amounts of resources and infrastructure required to operate a Big Data application, the bigger companies have the majority share of applications [31].

Since Hadoop came out of Yahoo, it is no surprise that Yahoo has had early success with Big Data infrastructure. The Yahoo web search runs on a Hadoop cluster of 10'000 cores. To reflect what this computation power is like, Yahoo was able to sort a terabyte of data in 62 seconds over 1460 nodes [32]. Facebook is also a big player in the Big Data space, boasting 200 terabytes of storage in 2010 [33]. Reddit also uses a Cassandra database as their storage solution[34].

In the telecom sector, Big Data is used to handle the large amount of communications and network data that is generated. Cisco, uses Cassandra on their WebEx platform to store user feeds and activity data [35]. Moreover, British Telecom (BT) offer Hadoop based Big Data services that can monitor web-based security threats in real time [36].

## **Chapter 3**

# **Project Requirements**

This chapter will provide context for the project from an industry perspective and outline where the project will be used in practise. The main objective of this chapter is to formalise a specification of deliverables to benchmark the final outcome against. Furthermore, there will be a brief discussion on how the outcomes will influence the wider project, both from a software engineering standpoint as well as a research objective.

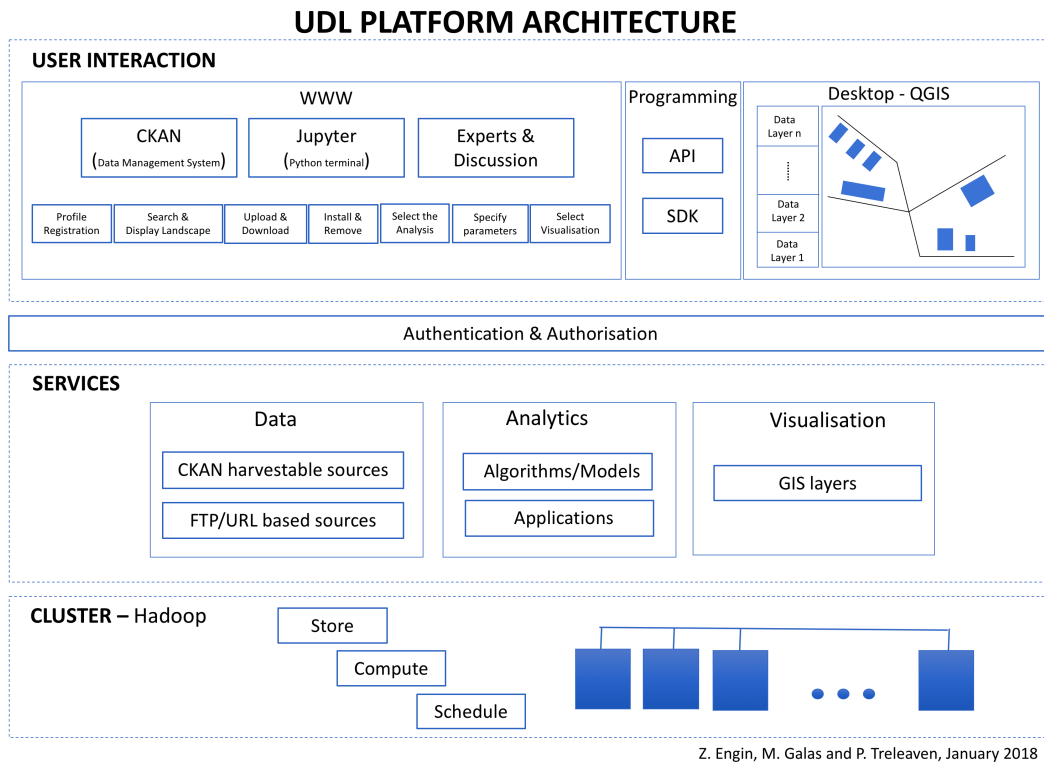
### **3.1 The Urban Dynamics Lab**

The main benefactor of this project, certainly from a software engineering standpoint, will be the Urban Dynamics Lab (UDL.) It is a research project funded by the UK Regions Digital Research Facility and aims to combine the expertise from several existing University College London (UCL) research groups such as Space Syntax, Centre for Advanced Spatial Analysis, the Geography department and the Computer Science department. With the combined expertise, UDL looks to explore the potential of city and economic development by using urban modelling, data analytic and computing.

It also worth noting the support the UDL has from both research organisations and industry. Partners include the Bank of England, Greater London Authority, SAS, Intel, Future Cities, Core Cities, Space Syntax Limited and Centre for Cities.

### **3.2 Platform Overview**

One of the main interdisciplinary projects that UDL is currently running is a large Big Data driven computing platform that is accessible to urban planners, geographers and data analysts. The platform consists of various services that enables easy access to datasets and provides faster data modelling tools. Although end users are expected to have basic coding experience, mostly in Python or R, they are not con-



**Figure 3.1:** Overview of UDL’s entire data analytics platform

sidered skilled enough to write Hadoop MapReduce algorithms or Spark programs to leverage distributed computing power as specified in Chapter 2.

In Fig 3.1, the complete platform design is provided. There is a key distinction between the client-facing interfaces and the behind-the-scenes, ”backend” type systems. The core computing elements of the cluster is a cluster which uses the Hadoop Distributed File System (HDFS) to store urban data across the different physical nodes in the cluster. There are also other specific tools that are used by urban planners such as the CKAN plugin which makes visualising geo map data easy and simple.

As mentioned in the literature review, using the HDFS and performing computations using either MapReduce or Spark code is outside the scope of what most urban planners would be able to do. Therefore, the backend Hadoop system is a complete blackbox to them and a layer of abstraction is required to allow the end users to interact. This is the precise goal of the Jupyter Notebook web interface.

### 3.3 Jupyter Notebook web interface

The Jupyter Notebook interface is available to all end users and should be accessible through a web browser within the UCL network. The interface should look exactly

the same as a normal Jupyter Notebook however it should be running using a Spark kernel. This means that any code written in the Notebook will be compiled and executed as a Spark program using the computing cluster UDL has. This means that the average urban planner only needs to write Python code to be able to access the computational benefits of Hadoop and Spark.

### 3.4 A practical use case

Alongside the Jupyter Notebook interface, the project also includes a proof-of-concept that shows how the Jupyter Notebook interface should be used. The idea is to use the Jupyter Notebook as an urban planner would, to take different sets of urban data and use them to find some correlation between them to find to make smarter urban planning decisions or learn something new about the city.

The Jupyter Notebook interface will be used to answer the question, **”Is there a relationship between air pollution and construction sites in London?”** This is a question of interest to the UDL and there are certainly some interesting datasets which can be added to the UDL repository by exploring this question. There is also scope for including other external datasets to see if there are any issues that can be accessed, such as cycling data and can be used to ask further questions like, **”Do London cyclist cycle past places of a lot of air pollution or construction site?”**.

Whilst this remains an entirely technical exercise in proving the plausibility of the Jupyter Notebook interface, there can be some interesting results from the questions answered. The results can be used to address social, economic and environment policy around London which is one of the key goals of the UDL. It can also be used to hold councils accountable and be a showcase of the benefits of open data.

### 3.5 ETL Pipelines

As the proof of concept requires data, one of the subtasks will be to build pipelines that can extract, transform, normalise and save the data onto the platform’s HDFS. These ”pipelines” will be automated so that they can be scheduled to run whenever they administrator of the system decides is best.

Each dataset that will be looked at will be in a specific format and this project will work with the urban planners at the UDL to get it into a format that they are happy to work with, so that it reduces the amount of work they have to do when writing code or doing analysis.

## 3.6 Formal specification

To conclude, the project's key deliverables can be categorised as follows:

- **Jupyter Notebook interface** - the end user facing Jupyter Notebook service will have a Spark kernel driver installed allowing it to make fast computations given Python code.
- **Expose the Jupyter Notebook service** - users should be able to access the service through the UCL network, Eduroam, by logging in with a password for authentication. (Note that more advanced authentication is beyond the scope for this project)
- **Air Pollution Pipeline** - this pipeline should extract and transform data from sources of air pollution data in order to normalise and store it for urban planning usage.
- **Construction work Pipeline** - this pipeline should extract and transform data from sources of construction work, happening particular around London and store it for future use.
- **Automated pipeline scheduling** - use a open source scheduling framework to run the pipelines aforementioned to automate data retrieval of both sets of data.
- **Evaluation of extracted data** - by using the built tools (the Jupyter Notebook and the pipelines), the data will be analysed to see if there are any key issues that arise or any conclusions that can be made by using the tools.

## 3.7 Software Methodology

The "Waterfall" approach was selected ahead of other commonly used software engineering practises to complete this project. This is because the benefactors (UDL) had a strict set of requirements which were established during the initial stages of project. As the goals were evident from the start, a strict plan was formulated to ensure that different components of both the deliverable and the final report were completed on-time. The initial planning also ensured sufficient time was given to each stage of the process.

The plan was visualised through the form of a Gantt chart as shown in Figure 3.2. The purple blocks represent dissertation related activity whilst the green blocks

Task	October	November	December	January	February	March	April
Research frameworks							
Research datasets							
Research literature							
[Disseration] Write abstract & introduction chapters							
Prototype with Jupyter Notebook & other frameworks that will be used							
[Disseration] Write abstract & introduction chapters Write literature review chapter							
[Disseration] Write abstract & introduction chapters Write requirements chapter							
Implement Notebook functionality							
Implement pipelines							
[Disseration] Write implementation chapter							
Testing & Documentation							
Write testing and conclusion chapters							
Review & improve dissertation							

**Figure 3.2:** The timeline of the project as a Gantt Chart.

are for system implementation. This project plan addresses all the deliverables addressed in the previous section and allocates enough time for each component.

Iterative software methodologies were also considered, but were found to be unsuitable or poorly suited for this project. For example, the "Agile" way of working is better suited for projects with more flexibility or require greater interactivity between different principals to evolve the product. This project had rather rigid goals and did not require much discussion or "evolution" through iterations.

## 3.8 Version Control

This project made use of the popular distributed version control software GitHub was used. Other teams working on the larger UDL project in the past have their projects on GitHub and so naturally this was an easy decision to arrive at.

However, the use of GitLab was considered as it provides free continous integration for private repositories, where as GitHub's Travis CI is a commerical tool. However, the advantage of this is not that relevant for the key project requirements and so GitHub was selected. Github repository link(s) are provided in Appendix A.



## Chapter 4

# Urban Planning Datasets

This chapter details the datasets that are used in this research project, specifying the origin, format and other attributes associated. The impact and potential uses for the data is also explored in the context urban design and planning, focusing on it's relevance. The goal of this chapter is to outline key problems associated with the datasets so that they can addressed through the ETL pipelines.

### 4.1 London Development Database

The London Development Database (LDD) holds all urban planning records in London since the year 2000 and is available from the British Government's open data website [37]. The records consist of descriptions of the type of work carried out, the status of the application as well as geographic details such as the location. This dataset is an example of a construction dataset which can be linked to another type of dataset (e.g air pollution) to help us visualise any correlations.

The format is quite unusual. It is available to the public through a web interface

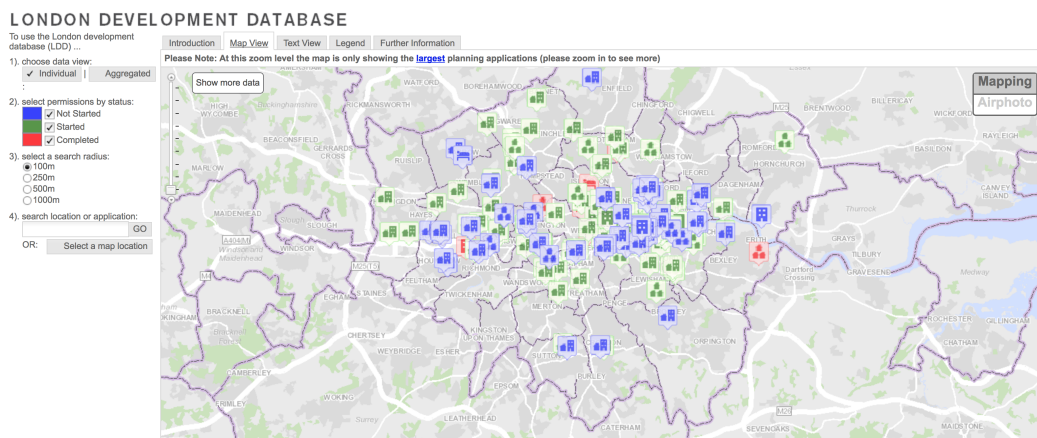
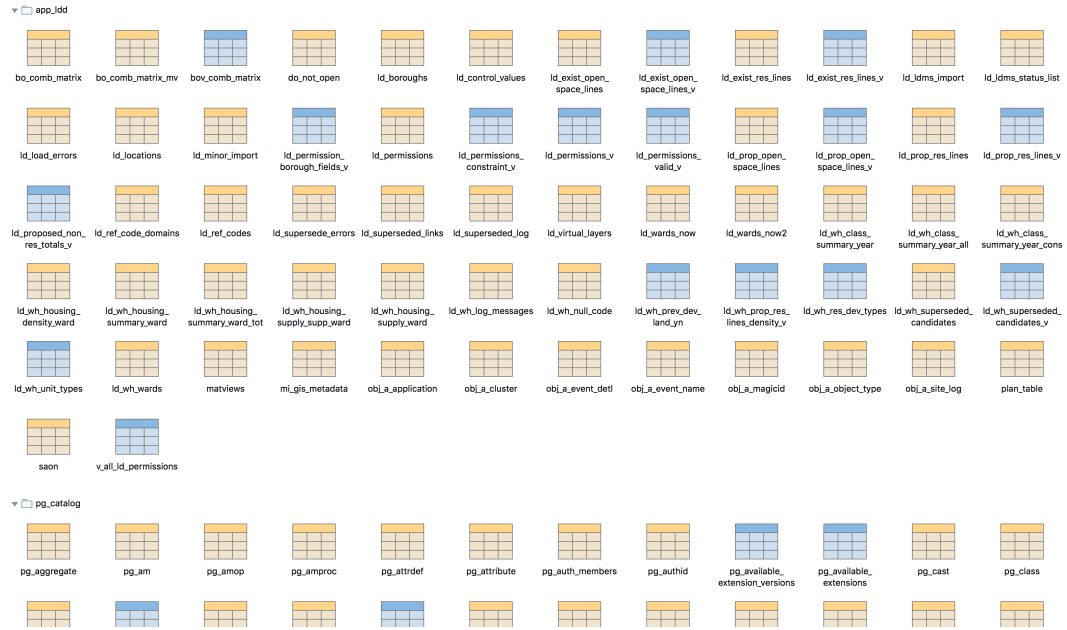


Figure 4.1: Public web interface for LDD



**Figure 4.2:** The many tables and views that exist in the LDD dump, most without documentation.

that allows for filtering and sorting queries on the dataset which is then shown on a map as shown in Figure 4.1. The raw dataset however is only available as Postgres database dump which utilises the Postgres SQL's pg dump method. Already, this imposes a constraint for most users who are used to CSV and spreadsheet based formats and in particular, urban design planners who would normally have difficulty accessing this kind of data.

Mounting the database manually requires the user to start a local Postgres server which then allows the user to query the dataset. However, there is an added complication that the documentation for this dump is minimal and there are also multiple views that exist without much of an explanation (Figure 4.2).

Ideally, the average urban planner would like access to the basic dataset perhaps with a few filtering options for ease of use. The format should definitely be a CSV as they are easy to work in the Jupyter Notebook set up. Furthermore, the dataset is fairly large and this would take a while to manually do, so an automated routine option would be suffice here.

One of the deliverables from this project therefore will be to build a pipeline which provide easy access to this dataset in an easy to use manner and in a simple CSV format for future analysis, both for this project and open sourced for others.

## 4.2 London Air Pollution API

The London Air pollution API provides access to pollution levels in and around the different boroughs of London. This data comes from the pollution stations that are situated at specific sites and are able to collect information on more than one type of pollution and at different intervals. The project is endorsed by the Mayor of London and is ran with the help of several London universities [38].

This dataset can be used in conjunction with LDD to see if there is a correlation between pollution levels and the density of construction sites. In London in particular, air pollution levels are at a high so this becomes a huge consideration for urban planners in making more eco-friendly sites and designs.

As with LDD, a public web interface is available, although it is limited to a certain amount of queries and does not return full information. However, there is a REST API which can be made use of.

The API design is somewhat strange and difficult to use. Firstly, the entire historic data is not available, so all the information comes from their servers which are incredibly slow. Secondly, the design is very restricted as it limits the amount of information accessible. In particular, the information is not available through one API call, which makes the API incredibly inexpressive. For example, the user first has to make a call to get all the site codes with the pollution meters. Then for each of those site codes, roughly 266 as of 2017, separate calls are made to get information for a particular year which becomes a very slow process, with the server speed the bottleneck in this scenario.

Another limitation is that the API responses don't follow best practises. For example, when asked for the records of a certain location, the API returns all the items as a JSON list instead of dictionary or associative map. As a result, the developer is responsible for iterating the entire list just to find the object they desire. This poor designs makes scaling difficult and a lot of complex code is required to handle the edge cases of such a design pattern.

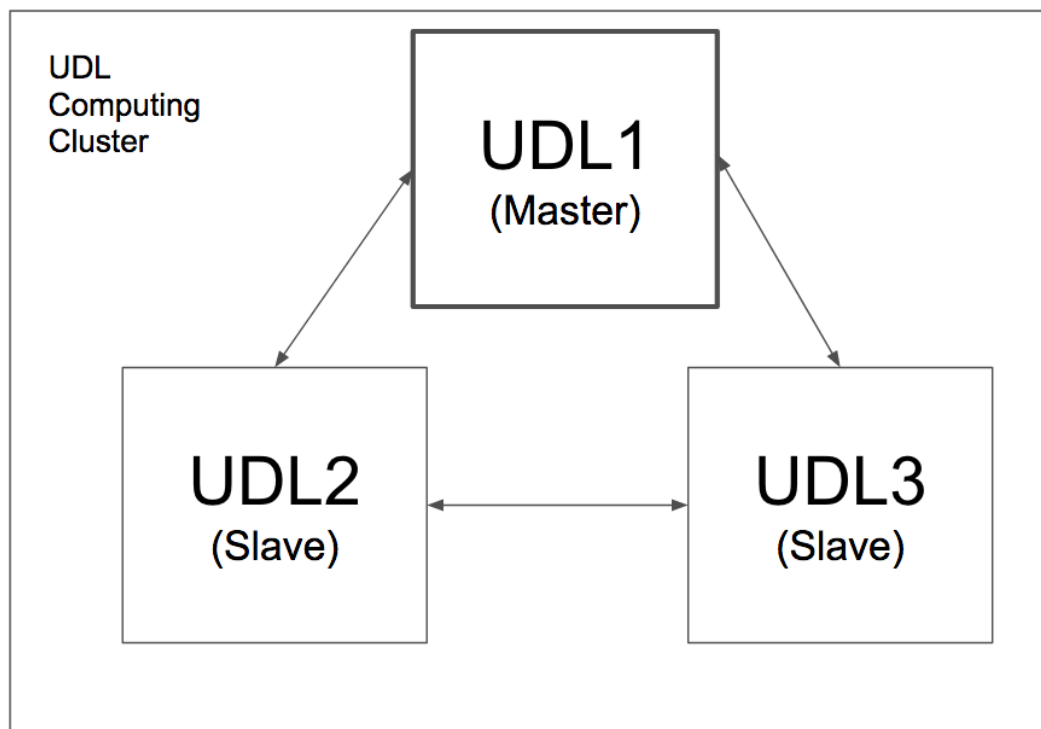
For an urban planner, what would be really efficient is that all this data is pre-populated and created as one single storage solution. This means that all the end points have been traversed to collect the maximum amount of information in advance. This can save times and avoid the server response time bottleneck. The historical data won't change either so this makes sense for a variety of reasons.

As with LDD, a pipeline that does some pre computation on the dataset would be very useful and can be used by the Urban Design Lab for future work as well as for this research project.

## Chapter 5

# System Implementation

This chapter gives a detailed overview of how the system was built. It also provides reasoning for the different design decisions that were made when implementing the key components of the deliverable. Finally, all system components are used together in a proof-of-concept urban planning task to demonstrate the final product's viability.



**Figure 5.1:** Node structure of the UDL cluster

## 5.1 Fixing the UDL Cluster

### 5.1.1 Outline of the UDL cluster

Let us first outline the structure and set up of the UDL cluster. It consists of three nodes: 1 master and 2 slaves as shown in Fig 5.1. Big Data services like Spark, HBase etc are installed using the Cloudera Manager software, a closed source, proprietary solution to installing and managing clusters. It allows one to track, view and interact with services all from a UI which can be useful when there are so many services installed.

### 5.1.2 Problem

It is to be noted that the Big Data services are not the only services provided by the nodes. The master node in fact hosts several non-Big Data services like CKAN which uses Postgres, a Drupal style forum and other web services. One requirement of this project is to install Jupyter Notebook as a service on the master node.

This project took over the UDL cluster in a non-functional state. There was an error which stopped the start-up of all the Big Data services and the first part of this project was to get the cluster working as a whole. This was not planned for, but with so many groups also implementing services on the UDL platform, it was inevitable that something would eventually break.

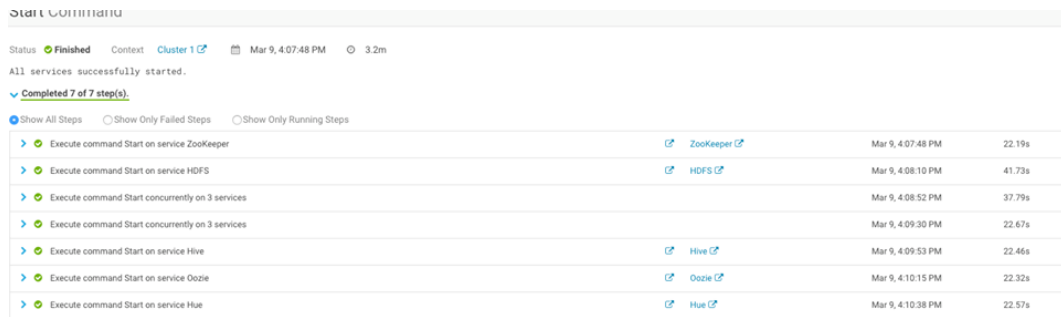
### 5.1.3 Solution

When Cloudera Manager is functioning correctly, it runs a service called *cloudera-scm-agent* on each node (including master). An additional service known as *cloudera-scm-server* which only runs on the master server. However, *cloudera-scm-agent* was not running on the master server - in fact the service did not exist at all on the machine. It's highly likely that it was deleted (accidentally) when another group was installing/modifying another service. The cluster was, in the end, restored back to full health. Fig 5.2 shows the start-up execution of all the services on all the clusters.

## 5.2 The London Development Database Pipeline

### 5.2.1 Existing problems

As described in the datasets, the London Development Database is a 2GB PostgreSQL dump which makes it difficult for urban planners to work with and extract information from. This format can even be a hassle for developers to initially start work with.



Step	Status	Description	Service	Timestamp	Duration
1	Completed	Execute command Start on service ZooKeeper	ZooKeeper	Mar 9, 4:07:48 PM	22.19s
2	Completed	Execute command Start on service HDFS	HDFS	Mar 9, 4:08:10 PM	41.73s
3	Completed	Execute command Start concurrently on 3 services		Mar 9, 4:08:52 PM	37.79s
4	Completed	Execute command Start concurrently on 3 services		Mar 9, 4:09:30 PM	22.67s
5	Completed	Execute command Start on service Hive	Hive	Mar 9, 4:09:53 PM	22.46s
6	Completed	Execute command Start on service Oozie	Oozie	Mar 9, 4:10:15 PM	22.32s
7	Completed	Execute command Start on service Hue	Hue	Mar 9, 4:10:38 PM	22.57s

**Figure 5.2:** All running services on the cluster

One of the ways in which database can be used is to create a local Postgres server and then to rebuild the database in that server. This can be a difficult task for urban planners but it's also incredibly unnecessary to start a local server just for a single database. A better approach would be to have a dedicated Postgres server that holds this data along with any other SQL databases.

However, this would require additional hardware and maintenance which is not of value for a single database. Another issue is that there are other Postgres services which exist on UDL as mentioned in the previous chapter. This means that dumping the database into an existing Postgres instance is not a viable option.

### 5.2.2 Solution: Docker

The use of virtualisation, through Docker, makes this tasks significantly easier. The user would only have to install Docker (a very easy installation compared to Postgres) and run a single script which starts a Docker container on whatever machine that the user is on, then mounts the entire datasets and exposes ports for the user to connect to. Docker also allows the user to dictate which ports should be opened and what the performance should be like without writing more code.

This approach is very suitable for the current Urban Design Lab (UDL) cluster. The cluster has several Ubuntu nodes linked together, so the Docker contain can run on any of those nodes and an arbitrary not-in-use port can be opened up.

In the end, the actual implementation for this is incredibly simple (although it did take a while to figure out.) Docker has a functionality called Docker-Compose which is essentially a build script for a container. We specify a container, in this case, Postgres. Note how this is just a simple Postgres installation, and not an OS-specific Postgres server (this makes it very lean.) We also provide the core SQL dump file and the build script which gets invoked when the container is run. Finally, there are some arbitrary options such as username, password and port mapping options.

**Listing 5.1:** The simple Docker file

```
postgres:
  restart: always
  image: postgres:10.1
  ports:
    - '1111:1111'
  environment:
    POSTGRES_USER: 'london_user'
    POSTGRES_PASSWORD: '***'
  volumes:
    - /abs_path_to_files:/docker-entrypoint-initdb.d/
```

With Docker, the Postgres database server is containerised to be able to extract all the information that is required and saved as an CSV file for the urban planner to use. The Docker container can also be used at any time to extract more information from the dataset. With this pipeline, the task of getting the data has become incredibly easier as the user doesn't have to worry about the OS-specific Postgres installation or setting up the server and can get to work in using the actual data a lot faster.

## 5.3 The London Air Pollution Pipeline

### 5.3.1 Existing problems

The issues surrounding The London Air Pollution Pipeline have been explained in chapter 4. The dataset is in a slightly easier format compared to the London Development Datastore and is available through a REST API. However, the design of the API endpoints and their response does not follow standards.

To reiterate some key issues, latency is a major problem with the API as it is extremely slow. Furthermore, there is no option to send bulk requests with a single REST call. For example, to find information for a given site, the user must make individual calls to the API for each site, which with its latency becomes an incredibly slow process.

The goal with the London Air Pollution pipeline is to automate this querying and allow this to be scheduled in advance rather than executed when the data is required. This allows urban planners to spend more time working on their dataset, rather than waiting for it to load.

### 5.3.2 Suitable programming languages

The pipeline was written in Java 8 and was chosen over Scala for simplicity. Scala is great when working with other Big Data technologies like HBase as it allows the user to fully take advantage of Scala's functional programming and implicit type casting features, none of which would be useful in this scenario.

Python was another alternative that was considered and it provides the great benefit of fast prototyping, easy HTTP calls and easy support for handling JSON data. However, Python is also slower and provides less control for the developer. Moreover, Oozie (the task scheduler in use) is used primarily to run Java pipelines so this would be a suitable time to use Java to test the proof of concept.

### 5.3.3 Pipeline walk through

We take advantage of the simple HTTP requests library and Google's JSON library, GSON. Writing networking code in Java can be quite verbose, the simple HTTP library makes this process very abstract for us as we only need to make GET calls. Support for JSON in Java is also not native as Python and so this library comes in very useful.

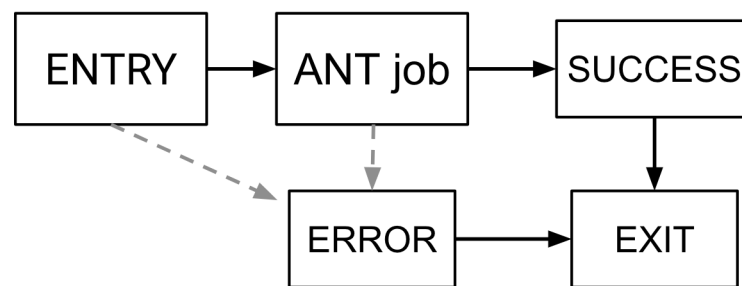
The pipeline begins by collecting a list of locations where the pollution measuring stations are located, these are known as site codes. Once the list has been retrieved, queries are built for each site code and for every year within a range from 2000-2018. Not all site codes have data for certain years and there is no easy way of retrieving this without querying for every single year. Naturally, this is an exhaustive approach, running in quadratic time complexity and with the latency problems described earlier, it becomes very slow.

The results of the query (site code + year) is then converted into a JSON representation. It is then saved into the file system and separated into folders by year and site code.

There should be some mention of the serialising and de-serialising process at work here when working with JSON in Java. Usually, given some raw JSON text, one can usually convert it into a native Java object if the names and fields match. However, the API conforms to a naming system that prepends its JSON keys with '@' and Java variables can't start with the '@' character. As a result, a custom serialising and de-serialising interface has to be built for every JSON result type as Java does not accept variables with special characters. Whilst this is easier to deal with Python, it also provides more formal control over naming conventions when switching between raw JSON and native object representation.

The Java project uses ANT as a build tool as it's easy to use for simple projects





**Figure 5.3:** DAG for the Air Pollution pipeline

like this. It's easy to manage library dependencies, simply include the JAR files in a folder. This makes linking it together easy.

In conclusion, this pipeline is able to robustly provide access to data and can be scheduled using Apache Oozie. By running this pipeline once, the urban planner will have access to the data in advance and can start analysing it quite quickly. Full Javadoc style documentation is provided for all classes used in this Java project as well as the UML which diagram can be found in Appendix D.

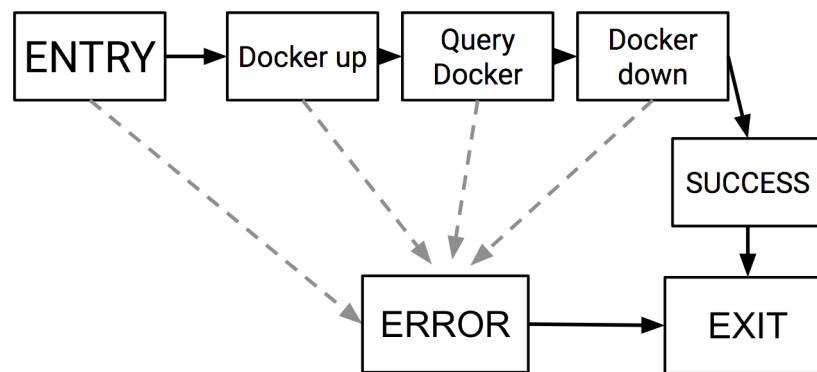
## 5.4 Automated Pipelines with Oozie

Apache Oozie is an open source scheduling framework that's based of dependency based direct acyclic graphs (DAG). It's very useful in orchestrating the running of multiple Big Data applications (such as mixing Hive code and MapReduce code). Amongst its many uses, it allows us to run pipelines at a pre specified time.

For the pipelines built (Air pollution and the London Data Store), we simply define two Oozie tasks to run on the server. Whilst these can be run automatically, we'll set manual triggers for now (UDL administrators can configure this later.) There are no build dependencies for either of the pipelines so it can be run as is.

High level workflows are presented in Figures 5.3 and 5.4. With the Air Pollution API, since most of the tasks of building and executing the project are handled by ANT, we simply execute a bash script that calls ANT in the appropriate folder. Meanwhile, with the London Development Datastore pipeline, there are stages involves in starting and using a Docker instance and using it to gather data.

In all cases, there are no specific error responses, so all the errors are caught (indicated by the grey line) and the program is executed. This can be added upon in the future should the stakeholders chose to do something interesting with it.



**Figure 5.4:** DAG for the London Development pipeline

## 5.5 Jupyter Notebook with Spark Integration

One of the core features of this project is to allow Jupyter Notebook applications to run using a Spark driver. This means that all operations are Spark native and thus work faster on the large sets of data than if they were to be run locally.

Before starting the installation, we must set up a dedicated user as this follows best practises. A new user is created the installation files will be placed in the new user's home directory. This will make things easier to manage when handing over. Creating the user is a trivial step,

The installation of the driver itself is fairly straight forward. First, we begin by creating a new "Virtual Environment" using the virtualenv library. This is a really powerful and useful tool because it allows us to separate machine-wide configurations and dependencies and those needed for our build of Jupyter Notebook. This is rather essential on a computer cluster like this where many libraries are installed and are used for different purposes, so it is best to have a separate environment.

Since Jupyter Notebook uses a Python environment, we can also install libraries that are often used in data analysis like Pandas, Scikit Learn, etc. These are again installed locally to the Python version which exists in the virtual environment so frameworks versions can be easily separated. These libraries can easily be added or removed by updating the requirements.txt file.

Once Jupyter Notebook has been installed, we can proceed to install the Spark driver. This is done using an open source framework called Apache Toffee, which provides a Jupyter-Notebook/Spark driver. The only parameter required is the path to our Spark framework binary folder. (There are also some Unix specific file ownership issues which one must handle but that is very minute detail.) Once the driver is installed, we can configure Jupyter Notebook as normal. As this uses the binary of the Cloudera Spark installation, it means that it can automatically access all the

nodes on the cluster through a single Jupyter Notebook

Next, Jupyter Notebook needs to be configured as a service. we need to specify up the port that the service can be accessed through. There are also some authorisation parameters, host, browser settings and password settings which need to be configured. The service itself is run using the systemd functionality on Ubuntu. The config file is provided:

**Listing 5.2:** Systemd configuration file for Jupyter Notebook service

```
[ Unit ]
Description=JupyterNotebook

[ Service ]
Type=simple
Environment="PATH=$env_paths"
ExecStart=/home/jnserver/jn/env/bin/jupyter-notebook
User=jnserver
Group=jnserver
WorkingDirectory=/home/jnserver/jn/notebooks

[ Install ]
WantedBy=multi-user.target
```

The key elements of the configuration file are the *User* and *Group* field which we set as the user we created. The *Environment* is set to the virtual environment we created. Finally, *ExecStart* points to the binary file of Jupyter Notebook inside the virtual environment. This is a tidy contained installation and adheres to best practises.

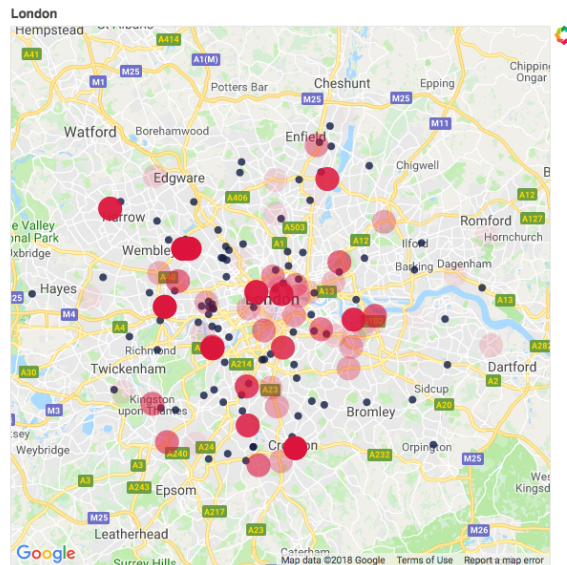
To conclude the installation, we set the systemd service to start on machine start up so it can be accessed as long as the server is turned on.

## 5.6 Bringing it all together: A practical case-study

With the installation of the Jupyter Notebook interface complete and the pipeline also finished, we can use both of these tools to answer the proposed urban planning question as specified before.

All of this done on the Jupyter Notebook interface so we can simply access the URL e.g (www.udltest1.cs.ucl.ac.uk:4567) it will give us the Jupyter Notebook interface and we can start writing Python code. The entire Notebook is provided in the code that is submitted, but it's also worth going over some of the key ways in

```
In [68]: # Finally plot the Pollution Data
pollution_NO = Circle(x="lon", y="lat", size=25, fill_color="crimson", fill_alpha="site_val", line_color=None)
plot.add_glyph(pollution_site, pollution_NO)
show(plot)
```



**Figure 5.5:** Final visual representation of construction site data against air pollution

which it works.

Firstly, the libraries are imported. Due to the way the systemd service is set up, it's easy to install new libraries as they are all hosted in the environment folder. From this single directory, libraries can be installed into it and provided to all users of the service.

Then we load both the datasets and attempt to normalise them. For the purposes of this case study, we will be mapping the datapoints on to the graph. Thanks to the TFL Air Pollution pipeline, it automatically provides geo coordinates which aren't normally available through normal API calls (hence the benefit of this pipeline). For the LDD dataset, we need to get the geoco-ordinates which isn't too difficult and could probably be another pipeline in the future. There are other data analytics that is performed on the end points to normalise it which can be found in the source code but is of no value exploring here.

Finally the data gets plotted on top of the map of London. The pollution values are indicated by shades of red - the darker, the more polluted - whilst the construction sites appear as dark blue dots. In these simple steps, we are able to provide an interface that completes the task and can answer the question. The results from this will be analysed in the next chapter.

## Chapter 6

# Testing & Results

This chapter covers the testing that was conducted on the software that was produced as a result of this project. This is followed with a presentation of the results and a discussion on the wider implications with respect to the questions proposed in Chapter 3 as well as the larger goals of this project.

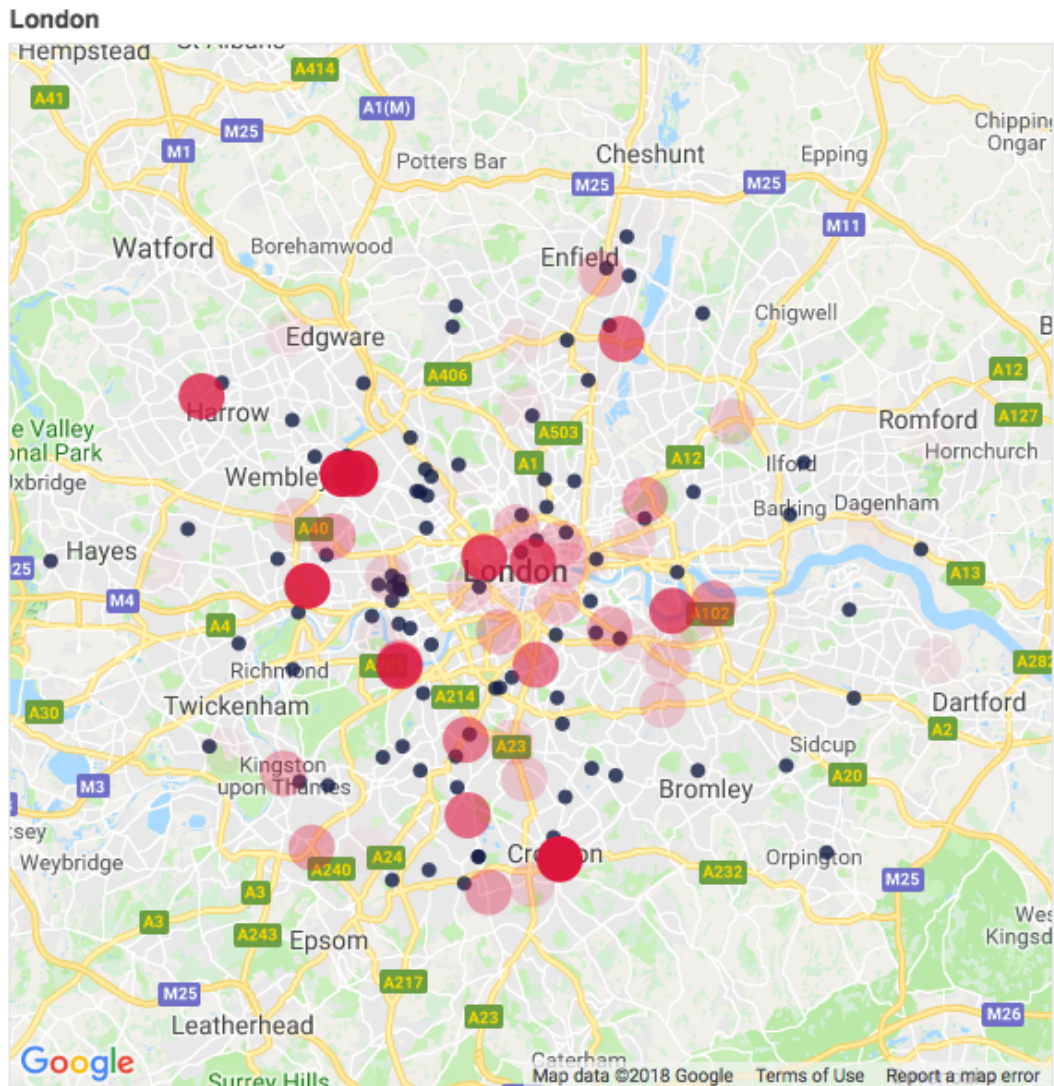
### 6.1 Answering an urban planning question

In Fig 6.1 the final map of the two datasets overlaid on top of each other is presented. The method and technique used to create this is described in the previous chapter. Note that this data concerns data points from 2015 only and considers the pollution indicator N02. Through the Jupyter Notebook service and the pipelines created as part of this project, the urban planner is able to quickly put together an overlay of two geo-spatial datasets and apply their domain-specific knowledge to dataset for quick prototyping and modelling.

The red circles represent the pollution data and their opacity indicates the how high the pollution levels are. The dark blue dots represent construction sites. We see that there are high levels of pollution in central London (as expected). However, there is also a high level of pollution in North East London (near Wembley) as well as Croydon which is slightly more unexpected. Furthermore, the map also tells us that there is a lot of work happening in West London, around Ealing which is likely to be due to the new Elizabeth tube line being installed.

There isn't as direct link between air pollution and constructions sites as hypothesised since the Ealing area has little pollution but many construction sites. The opposite is true for central London with high levels of pollution and few construction sites. This may suggest that air pollution is caused more by traffic than construction sites.

The urban planner can further investigate the relationship between air pollution



**Figure 6.1:** Final map of overlaid dataests

and construction in the larger scheme of the other UDL datasets available. There are many other factors which can affect this relationship and broader look into all the years and all the pollution types need to be considered before arriving to a conclusion.

However, with the tools provided in the project, this is certainly doable as urban planners have the data easily accessible to them. This prototype acts as evidence of what is possible with the interface and pipelines provided and so can be considered a success.

## 6.2 The Jupyter Notebook Service

The main requirement of the project was to implement a Jupyter Notebook service for the UDL data analytics platform, alongside a Spark driver which lets the user take advantage of the UDL computing cluster.

### 6.2.1 Build Testing with Docker

It is difficult to test the Jupyter Notebook service or any other service like this as there are no simple input/output expectations. One cannot simply write unit tests either.

However, it is viable to build test. This was done through the use of Docker. Docker allows a user to virtualise the target platform e.g Ubuntu and run a set of commands on the virtualised container. This ensures that the build test is free of any machine dependencies and Docker in general is easy.

The tests consists of starting up a Docker-Ubuntu instance and then running a build script. The build script essentially contains the set up instructions that were described in more detail in the implementation chapter.

Docker will progress through the build script and only report errors if the installation fails at any point. If the build passes, it returns a "0" code which implies that everything has installed as per the set up script.

This is a great way of ensuring that the build script is replaceable on many machines and is a good way to test the installation instructions of the Jupyter Notebook

### 6.2.2 Results

The Jupyter Notebook interface was implemented and works successfully. It's installation follows best practises and exists a proper SystemD service on the Ubuntu server under a dedicated user. The service has it's own dedicated environment for libraries which means that other users of the server will not be able to interfere with the installation. This means that this installation is safe and bound to continue to work in the future after this disseration has been handed in.

For the purposes of this project, the implementation of this service fulfils the project requirements in it's entirety.

## 6.3 The London Development Database Pipeline

The London Development Database pipeline is used to gather data from the London Development Datastore (LDD). The data from LDD is used to answer an urban planning question as specified in the project requirements.

### 6.3.1 Unit Testing with Nose

Since Docker was used in the database itself, it would be arbitrary to create a build test. Since the pipeline involves setting up a database, we can do some unit tests on the database to see if it works.

To accomplish this, Nose, a Python unit testing framework was used. Apart from being very easy to use, Nose is Python based and as we expect the users of the Jupyter Notebook service to use Python, it makes sense to use this a programming language to test the database with although any programming language could of been used for the tests.

To conduct the unit tests, the Docker instance with pre-loaded database is initialised. The ports are noted. Using Python, we create a series of SQL calls to the Docker end points. We test to see if the results are as expected. This is done for several scenarios (such as existing or non-existing columns) to ensure that the database works as intended.

One issue with testing this database is that it's difficult to formalise what is expected *exactly* from the database. This is largely due to the fact the documentation for the database is so poor so one can only stick to doing tests as they see fit.

### 6.3.2 Results

This pipeline is fully working and allows the user to extract data from the database without having to fully mount a Postgres instance on their machines. This makes it easy to use, particularly in situations where there are existing Postgres instances, like the UDL server. The pipeline allows the user to get the data they require without much difficulty so completes its goal.

## 6.4 The London Air Pollution Pipeline

Another pipeline that was created was the London Air Pollution pipeline. It aims to make it easier and faster to get Air Pollution data as it periodically.

### 6.4.1 Unit Testing with JUnit

Since this pipeline is written mostly in Java, JUnit was chosen as the unit testing framework. JUnit is extremely popular and widely used and integrates with the build tool in use (Ant).

### 6.4.2 Build Testing in Docker

Once the unit tests are run, we can also perform this on a target platform using Docker. We can quickly spin up a Ubuntu instance, mount the files and run ant.



Since Ant itself is a build tool, we simply call Ant in the project directory to run the code and the tests.

### **6.4.3 Results**

This pipeline works as required for the completion of the project. It simplifies the process of gathering pollution data in an automated process. It's also well tested and well documented so it can be continued to be used by UDL.

## Chapter 7

# Conclusion & Future Works

This chapter summarises the outcomes of this project and the progress made relative to the requirements specified in Chapter 3. There is a critical evaluation of the final deliverable as well as the project as a whole. Finally, this chapter finishes by suggesting future directions that this project can be taken.

### 7.1 Summary

This project has contributed new functionality to the Urban Dynamics Labs' (UDL) data analytics platform which will help urban planners and other experts investigate urban data in a simpler and more efficient way.

The key contribution of this project is the Jupyter Notebook interface with a Spark-driver enabling users to use the UDL Computer cluster through the interface. This in contrast to writing Hadoop MapReduce or Apache Spark code which is verbose and overly complex.

The interface has been tested by proposing an urban planning related question, "Is there a correlation between air pollution and construction sites in the UK?" This would be of particular value to urban planners who are looking at locations to create new buildings or would like to further explore the air pollution in London.

In order to answer this question, two extract, transform and load (ETL) pipelines were created to make it easier for urban planners to access the data in question. The first drew upon the London Development Datastore which holds construction data for London and second was the Transport for London (TFL) Air Pollution API.

For the reasons explained in the datasets section, both datasets were found to be difficult to work with or were presented in awkward formats which would cause difficulty for urban planners to access.

This project contributes two pipelines that can be scheduled to run at a specified

time. This means that the data is readily available to the urban planners instead of waiting for data to download or them manually trying to access this data. These pipelines will go towards a library of pipelines used by the UDL to perform data transformation tasks and make analytics more efficient for their staff.

Finally, all of this was brought together through the Jupyter Notebook and some basic modelling was done to prove the viability of the system. The research question proposed in Chapter 4 was indeed answered, with some evidence to suggest that there is a higher than normal pollution rate around North East London. Furthermore, it was found that there is some correlation between construction sites and the pollution levels, but more sources would be required to be able to fully confirm this. The interface allows urban planners to do this which was the original intention, hence the goals of this project have been met.

## 7.2 Evaluation

This project meets all requirements as specified in Chapter 3 and the deliverables have been signed off by the project's technical supervisors. Furthermore, the project was completed on time and followed the provided time frames well.

Despite this, there could of been a larger scope in the requirements for creating more pipelines, thus testing the interface in a wider manner to document and provide more examples of how it can be used. There could of also been a more specific focus about the research question as it was never verified with an professional urban planner. With more time (and resources), parts of this project could of been done in conjunction with a field expert to come up with more applicable data to the industry.

One of the biggest challenges with this project was the integration of other services that were being developed for the UDL Big Data analytics platform by other students (some from the Systems Engineering course, others for their final year project). There was very little central co-ordination between the student teams who were experimenting and installing their services on the same UDL cluster.

This desynchronised way of working slowed down the progress of all the UDL teams as every week it would emerge that a team has lost a library or framework they had installed and it was difficult to work out who had committed that change.

Moreover, the Linux-centric administration was poorly handled as everybody implementing a service for UDL was given root access. This caused all sorts of confusion about how file privileges and again, contributed to the issue of overwriting each other's work.

To address this, a suggestion has been made to the technical supervisors to have a dedicated system administrator or architect who is fully aware of the file

locations and where the libraries reside to make it easier for teams to implement their services. This could also be supplemented with a communication platform like Slack to remain in touch when not discussing the state of the platform during weekly meetings.

## 7.3 Future Work

There are many directions this project can be taken in the future. One area which needs to be addressed is the security aspects of this system. If this interface is intended to be used by many analysts then there has to be some way of constraining each user so that they do not overwrite each other's work or delete files etc.

This feature was discussed before starting the project and was adjudged that it would form a new project later on. This desired security mechanism would need to encompass the wider services as part of UDL and a more extensive approach needs to be taken in implementing a security access mechanism for the services.

Moreover, one could also test the interface with urban planners and get feedback from researchers regarding the usability through a case study. It was not required in this project as it was deemed out of scope. It would also require the time of a researcher which is a costly resource. The initial project plan as provided in Appendix B does include this task, however, this was dropped from the final requirements due to lack of resources. By doing this, one could provide empirical evidence regarding the usability of this interface and compare it with other interfaces.

Finally, one area that this project was not able to address was using this interface with other Big Data services like HBase or Hive. This was mostly because although the services are running on the UDL's test servers, they carry no actual valuable data that can be easily tested. Whilst the current Jupyter Notebook has the ability to tap into these services, more could be done to provide examples on how to use HBase or Hive. For example, another project within the UDL group was looking at visualising CKAN data from HBase - a possible extension could be to use the Jupyter Notebook to visualise CKAN data when there is usable data in these databases.

## **Appendix A**

# **Source Code**

All source code for the project can be found at this GitHub repository.

The repository where this project will be merged to can be found here.

## Appendix B

# Project Plan

### Vicky Dineshchandra - Final Year Project Plan

**Project Title:** Big Data analytics infrastructure for better urban planning

**Supervisors:** Dr Philip Treleaven , Dr Zeynep Engin, Dr Michal Galas

**Aims:** To build a Big Data infrastructure platform that allows the Urban Design Lab to analyse urban planning data through an alternative interface such as Jupyter Notebook.

**Objectives:** Research existing Big Data technologies to understand state of the art technologies that are currently in use

Research alternatives tool to popular Big Data technologies or similar products

Understand urban planning data sources and reasons for analysis Implement a Jupyter Notebook interface on a Hadoop+Spark cluster Test interface with Urban Planners

#### **Deliverables**

Literature survey examining Big Data technology, its origin and possible future directions A Hadoop/Spark cluster with a Jupyter Notebook interface A survey on urban planners testing the success of the interface Usability metrics Extra functionality such as security features and Continuous Integration Fully documented source and build code

#### **Work Plan**

Note that I plan on working on my dissertation through the year with the guidance of Dr Treleaven Project start to end of October

- 4 Weeks: Research + finalising project details
- November 4 weeks: Familiarity with Hadoop infrastructure + basic tests
- December 4 weeks: Literature review section of dissertation
- January 4 weeks: Experiment with Jupyter Integration

- February 4 weeks: Run tests and add CI
- March 4 weeks: Complete survey on urban planners
- April 4 weeks: Document all source code and tools used & polish dissertation

## Appendix C

# Interim report

### Interim Report

**Name:** Vicky Dineshchandra

**Project Title:** Big Data analytics infrastructure for better urban planning

**Supervisors:** Dr Philip Treleaven (main), Dr Zeynep Engin, Dr Michal Galas

**Progress made to date**

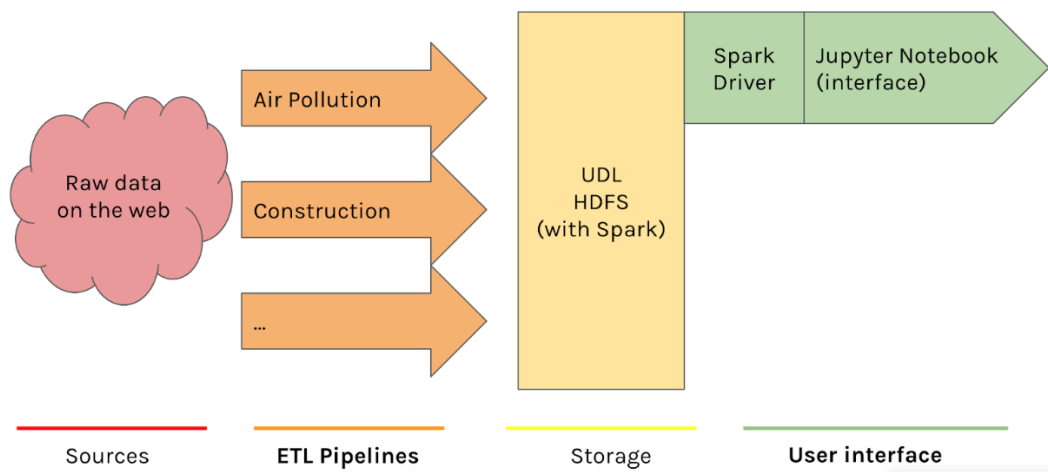
Ive become incredibly familiar with the current build of the Urban Design Lab computing cluster. Ive set up the Big Data Infrastructure system to use a Jupyter Notebook service accessible from the UCL network that requires authentication. The service runs a Scala kernel which connects to the clusters Apache Spark installation leveraging high speed computation power to the user of the Notebook. Port forwarding allows the service to be accessed from outside the cluster itself.

To test the Jupyter Notebook, we are looking to answer a question that an urban developer may want to ask in practise. To do this, Ive research and identified the appropriate datasets and worked on getting data out of them (the datasets are ridiculously cumbersome to work with and require a lot of development time to access). Some of the preprocessing code Ive written is of general interest to the Urban Design Labs as the data is something that they would like to use.

Using the datasets, the Jupyter Notebook service aforementioned can be used to cross-reference the data and build a map of air pollution vs spots of construction work. Although there are limitation, this is a proof of concept that this service works.

On the dissertation side, Ive almost completed my background section (still waiting on more specific research paper suggestions from the supervisors) and the datasets section. Ive been focusing more on doing the actual implementation recently, so it should be quite easy to write up.





## **Appendix D**

# **UML Diagram for AirPollution API**

UML diagram for the AirPollution API can be found [here](#).

Remaining work to be done before the final report deadline

More testing with the Scala/Spark kernel is required, on larger datasets Continuous integration testing + unit tests for all code Convert current code base to Scala, especially for preprocessing steps Create formal pipelines that run off Oozie, a scheduling engine Add more variation the current Jupyter Notebook experiments Design more experiments to test the Jupyter Notebook service with existing GIS tools that are used in industry Review non-technical papers focusing on the social / global / economic / geographical impact of urban design planning

# Bibliography

- [1] Big data facts. <https://www.forbes.com/sites/bernardmarr/2015/09/30/big-data-20-mind-boggling-facts-everyone-must-read>. Accessed: 2017-12-01.
- [2] Waterford big data stats. <https://www.waterfordtechnologies.com/big-data-interesting-facts/>. Accessed: 2017-12-01.
- [3] Mckinsey big data study. <https://www.mckinsey.com/business-functions/digital-mckinsey/our-insights/big-data-the-next-frontier-for-innovation>. Accessed: 2017-12-01.
- [4] Urban dynamics lab. <http://www.ucl.ac.uk/urban-dynamics-lab>. Accessed: 2017-12-01.
- [5] Big data in popular media. <https://www.swc.com/blog/business-intelligence/behind-buzzword-big-data>. Accessed: 2017-12-01.
- [6] Traditional data storages. <https://www.promptcloud.com/blog/big-data-handling-best-practices>. Accessed: 2017-12-01.
- [7] Harshawardhan S Bhosale and Devendra P Gadekar. A review paper on big data and hadoop. *International Journal of Scientific and Research Publications*, 4(10):1–7, 2014.
- [8] 5 vs of big data. <https://www.xsnet.com/blog/updated-for-2017-the-vs-of-big-data-velocity-volume-value-variety-and-veracity/>. Accessed: 2017-12-01.
- [9] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. *The Google file system*, volume 37. ACM, 2003.

- [10] About apache hadoop. <http://hadoop.apache.org>. Accessed: 2017-12-01.
- [11] John L Hennessy and David A Patterson. *Computer architecture: a quantitative approach*. Elsevier, 2011.
- [12] Hung-chih Yang, Ali Dasdan, Ruey-Lung Hsiao, and D Stott Parker. Map-reduce-merge: simplified relational data processing on large clusters. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 1029–1040. ACM, 2007.
- [13] Aditya B Patel, Manashvi Birla, and Ushma Nair. Addressing big data problem using hadoop and map reduce. In *Engineering (NUICONE), 2012 Nirma University International Conference on*, pages 1–5. IEEE, 2012.
- [14] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. *HotCloud*, 10(10-10):95, 2010.
- [15] Apache spark. <https://spark.apache.org/>. Accessed: 2017-12-01.
- [16] Philip A Bernstein, Nathan Goodman, Eugene Wong, Christopher L Reeve, and James B Rothnie Jr. Query processing in a system for distributed databases (sdd-1). *ACM Transactions on Database Systems (TODS)*, 6(4):602–625, 1981.
- [17] Stefano Ceri and Giuseppe Pelagatti. *Distributed databases principles and systems*. McGraw-Hill, Inc., 1984.
- [18] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C Hsieh, Deborah A Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E Gruber. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26(2):4, 2008.
- [19] Apache hbase. <https://hbase.apache.org/>. Accessed: 2017-12-01.
- [20] Lars George. *HBase: the definitive guide: random access to your planet-size data*. ” O’Reilly Media, Inc.”, 2011.
- [21] Apache hbase. <https://cassandra.apache.org/>. Accessed: 2017-12-01.

- [22] Divyakant Agrawal, Sudipto Das, and Amr El Abbadi. Big data and cloud computing: current state and future opportunities. In *Proceedings of the 14th International Conference on Extending Database Technology*, pages 530–533. ACM, 2011.
- [23] Sugam Sharma. An extended classification and comparison of nosql big data models. *arXiv preprint arXiv:1509.08035*, 2015.
- [24] Apache impala. <https://Impala.apache.org/>. Accessed: 2017-12-01.
- [25] Apache phoenix. <https://phoenix.apache.org/>. Accessed: 2017-12-01.
- [26] Apache spark streaming. <https://spark.apache.org/streaming/>. Accessed: 2017-12-01.
- [27] Apache kafka. <https://kafka.apache.org/>. Accessed: 2017-12-01.
- [28] Guozhang Wang, Joel Koshy, Sriram Subramanian, Kartik Paramasivam, Mammad Zadeh, Neha Narkhede, Jun Rao, Jay Kreps, and Joe Stein. Building a replicated logging system with apache kafka. *Proceedings of the VLDB Endowment*, 8(12):1654–1655, 2015.
- [29] Apache oozie. <https://oozie.apache.org/>. Accessed: 2017-12-01.
- [30] Finn V Jensen. *An introduction to Bayesian networks*, volume 210. UCL press London, 1996.
- [31] Big data cost. <http://blog.atscale.com/big-data-cost>. Accessed: 2017-12-01.
- [32] Owen OMalley. Terabyte sort on apache hadoop. 2008.
- [33] Big data at facebook. <http://blog.atscale.com/big-data-cost>. Accessed: 2017-12-01.
- [34] Big data at reddit. <http://highscalability.com/blog/2013/8/26/reddit-lessons-learned-from-mistakes-made-scaling-to-1-billi.html>. Accessed: 2017-12-01.

- [35] Big data at cisco. [https://www.cisco.com/c/en/us/solutions/collateral/enterprise/cisco-on-cisco/BigData\\_Case\\_Study-1.html](https://www.cisco.com/c/en/us/solutions/collateral/enterprise/cisco-on-cisco/BigData_Case_Study-1.html). Accessed: 2017-12-01.
- [36] Big data at bt. <https://www.globalservices.bt.com/btfederal/en/products/big-data-professional-services>. Accessed: 2017-12-01.
- [37] London development datastore. <https://data.london.gov.uk/dataset/london-plan-opportunity-areas>. Accessed: 2017-12-01.
- [38] Airpollution api. <http://www.londonair.org.uk/LondonAir/API/>. Accessed: 2017-12-01.