Evaluation of Hadoop Distributions for Data Science project

Abstract

This paper shows a general conducted comparison of Hadoop distributions which are dominating the big data market. Since it is necessary for our data science project to compute in a parallel and efficient fashion, such a comparison helps us to identify the right distribution. Firstly, an overview of the identified distributions with its features and pricing tiers will be showed. Afterwards I will declare some important criteria for our project based on an extensive research of internet blogs and manufacturer websites. Then a weighting of this criteria will be done. These criteria will be used to evaluate the different Hadoop distributions. The resulting evaluation matrix shows that HortonWorks might be the best distribution for our data science problem. However, a detailed look at the performance criterion shows us that MapR is the most performant in 5 out of 6 cases. For this performance comparison, evaluation results of various MapReduce jobs found on the Internet are used. Hence, in a pure performance comparison MapR would win over HortonWorks, but in an overall comparison HortonWorks convinces more. To round off the evaluation, a test implementation of HortonWorks is carried out on the 4 virtual machines procured for this purpose where some simple MapReduce jobs are carried out.

1. Hadoop Distributions Overview

For the first part of the evaluation, a search for widespread Hadoop distributions was conducted [1-5].

As the table below shows, 7 noteworthy distributions were found.

Hadoop Distribution	Features	Pricing Gear	Hadoop Ecosystem
Apache Hadoop 3.1.1 hadoop.apache.org	• No extra tools • Blank installation of Apache Hadoop (HDFS, YARN and Hadoop Web UI)	Open Source 100%	Has to be installed manually [6]
Cloudera CDH (v 5.15.0) www.cloudera.com	• Oldest distribution • Very polished • Comes with good (and proprietary) tools to install and manage a Hadoop cluster: • Cloudera Manager for managing and monitoring clusters [7]. • Cloudera Search (Free-Text) on Hadoop [8]. • Apache Crunch framework for MapReduce Pipelines [8].	Freemium (Cloudera Manager require license). Also source code is not fully available and enterprise edition has 60 trial-day [9]. Final costs may depend on cluster size [4].	Accumulo, Flume, HBase, HCatalog, Hive, HttpFS, HUE, Impala, Kafka, KMS, Mahout, Oozie, Pig, Sentry, Snappy, Spark, Sqoop, Parquet, Whirr, ZooKeeper [7, 8]
HortonWorks Data Platform (HDP 3.0.1) www.hortonworks.com	• Newer distributions • Tracks Apache Hadoop closely • Comes with standard and open source tools for managing clusters: • Improved HDFS in HDP 2.0: automated failover with a hot standby and full stack resiliency [10]. • Ambari for managing and monitoring clusters. • Includes almost all Hadoop extensions from the Apache foundation [10].	Open Source, optional enterprise paid support [9]	Atlas, HBase, HDFS, Hive Metastore, HiveServer2, Hue, Spark, Kafka, Knox, Oozie, Ranger, Storm, WebHCat, YARN, SmartSense, ZooKeeper [11]
MapR 6.1 www.mapr.com	• Uses own HDFS (MapR-FS) • Integrates own database systems (MapR-DB seven times faster than HBase) • Mostly used for big big data projects • Free from Single Point of Failures • Offers Mirroring and Snapshotting • Might be the fastest Hadoop distribution (see section 4, p. x) • MapR supports backward compatibility across multiple version of projects • Supports a global event replication for streaming at IoT scale (MapR-ES)	Freemium [12]: Community Edition contains no high availability, no disaster recovery and no global replication for streaming data.	AsyncHBase, Cascading, Drill, Flume, HBase Client and MapR Database, Binary Tables,, Spark, HCatalog, Hive, HttpFS, Hue, Impala MapR Event Store For Apache Kafka Clients and Tools, Myriad, OpenStack, Manila, Oozie, Pig, Sentry,

Hadoop Distribution	Features	Pricing Gear	Hadoop Ecosystem
			Spark, Sqoop, MapR Object Store with S3-Compatible API [13]
Intel www.hadoop.intel.com	• Partnered with Cloudera [14] • Encryption support • Hardware acceleration added to some layers of stack to boost performance • Admin tools to deploy and manage Hadoop	Premium (90 day trial)	Offers same services as Cloudera.
Pivotal HD www.gopivotal.com	• Partnered with Hortonworks [14] • Fast SQL on Hadoop • Proprietary Software [15]: ○ Pivotal DataLoader ○ USS (external file system) ○ Spring Data ○ Pivotal ADS-HAWQ (parallel SQL query engine)	Premium	Offers same services as Hortonworks.
IBM Open Platform www.ibm.com	• Partnered with Hortonworks [14] • Proprietary tools: ○ IBM Big SQL allows concurrent process of Hive, HBase and Spark and other sources using a single database connection [14]. ○ IBM BigInsights v4.2 provides Text-Analytics module [16]. • Highly compatible to other IBM products.	Free for non-commercial purposes, optional enterprise paid support [17]	Offers same services as Hortonworks.

Table 1: Overview of Hadoop distributions

It is worth to mention that each Hadoop distribution in table 1 comes up with a minimum of services from the apache foundation: YARN, HDFS, Hive, Pig, HBase, Kafka, Storm, Mahout, HCatalog… But they may use a proprietary implementation of them e.g. MapR uses own HDFS instead of Apache HDFS implementation. It is also worth to note that a few cloud providers are offering Hadoop cluster services over their platforms. For instance, Microsoft Azure provides with HDInsight a full manageable Hadoop respectively Spark Cluster [18]. The user could profit from fast provisioning and also from less costs since no on-prem hardware cluster infrastructure is required. Therefore, Hadoop (or better Spark) by Cloud Computing is also a mentionable option for our data science project in the future, although it may be oversized at the moment.

Nevertheless, Intel, ERM/Pivotal and IBM partnered with Hortonworks or Cloudera (Table 1). These vendors therefore have the same extensions as their partners offer, but also can provide exclusive tools for users (e.g. IBM Big SQL [14]). If we take this into account, we find that there are mainly 3 different Hadoop distributions: Cloudera with CDH, Hortonworks with its HDP and MapR with its own HDFS.

In addition to the distributions shown in Table 1, there are many other Hadoop distributions such as Altiscale from SAP or Elastic MapReduce from Amazon. Both run only in a cloud environment. The Forrester Wave shows also the three global players Cloudera, HortonWorks and MapR (see figure 1). Since cloud solutions do not play a role for this use case, Microsoft, Google or Amazon will not be considered.

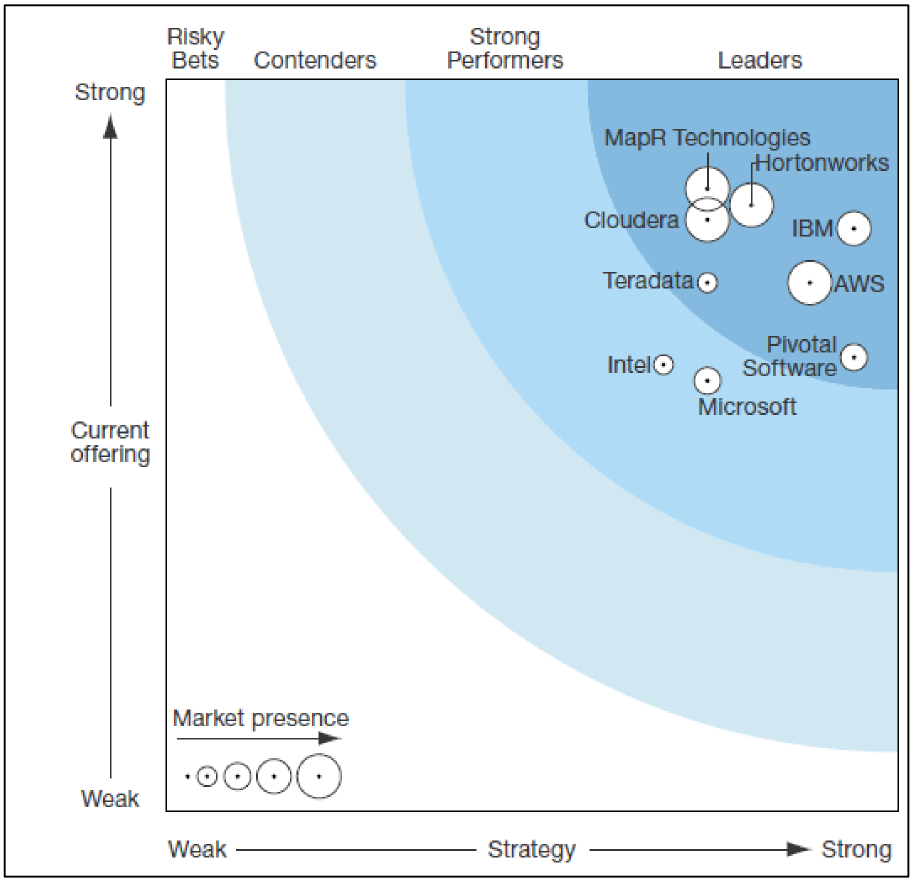


Figure 1: Forrester Wave of Hadoop Distributions [5]

As we can see from figure 1 the three mentioned Hadoop distributions are building a cluster and Cloudera even overlaps with MapR slightly. That means, they are providing similar products and enjoy a similar market position. From the Forrester Wave (figure 1) we can derive that MapR seems to have the best offering. However, since we want to use one of them for our analytics project we have to dig a step deeper and make a decision based on well-defined criteria.

2. Definition of criteria

Hadoop distributions are rated using selected criteria, with scores ranging from 1 (very poor) to 5 (very good). Based on the research carried out, the following 20 clear criteria can be determined:

Criterion	Description
Batch Processing	Batch process jobs can run without any end-user interaction or can be scheduled to start up on their own as resources permit.
Cloud Support	Cloud compatibility of Hadoop distributions.
Cluster Scalability	Creating new clusters ad-hoc and set up cluster size (e.g. 4,8 or 12 nodes).
Data querying possibilities	Ways to query Hadoop with SQL (e.g. Hive, Stinger, Spark SQL,…).
Database Support	Additional databases in Hadoop (e.g. PostgreSQL, MongoDB, HBase, Impala…).
Disaster-Discovery	There is a fallback option available in case of unexpected Hadoop errors.
Ease of Use	The complexity of the usage of the Hadoop distribution.
High-Availbility	Self-healing across multiple services or single failure recovery (i.e. fail-over clusters).
Install complexity	Is there a simple installation routine or does it require lot of manual configuration (e.g. Multinode Cluster…)?
Licensing and Pricing	Cost model of Hadoop distributions.
ML support	Providing interface for enlarged machine learning tasks (e.g. Spark MLlib…).
Operating System Support	Guaranteed and certified OS compatibility.
Performance	Cluster performance in case of parallelized MapReduce Jobs.
Programming Language Support	Support of data science programming languages (Python and R).
Streaming Analytics	Vendor offering real-time analytics capabilities (e.g. IoT platform hub..).
Support of secondary services	Services on top of Hadoop (e.g. Zookeeper, Spark…).
Third party module integration	Usage of additional components from other vendors
User Support/Community	How fast is the response in the community?
Expertise	Experience with Hadoop (life time of company, …)

Table 2: Overview of Hadoop distributions

The weight scale ranges from 1 (trivial) to 4 (very important), with the focus on our Santander Bicycle project. This means that cloud support, for example, is a crucial criterion in many use cases, but has only little importance for our project which leads to a trivial weight. A percentage weighting is not applied because 20 criteria would result in a fine-granular distribution (which is not good to read at all). Therefore, absolute values are used for the comparison table in the next section.

3. Weighted evaluation matrix

As already mentioned in Section 1, the three major Hadoop distributions are: Cloudera CDH 5.15.0, Hortonworks HDP 3.0.1 and MapR 6.1.0. With regard to the defined comparison criteria, a weighted decision matrix can be mapped. At the same time, this matrix represents the starting point for the decision of a Hadoop distribution. It cannot be denied that a certain degree of subjectivity is included in the evaluation (Table 2). In addition, only free editions are compared. For example, if a feature only exists in the premium version, this feature will be rated 1 (worst) since we don't want to waste money.

Criteria	Weight	KO	MapR	Hortonworks	Cloudera CDH
Batch Processing	3		5	5	5
Cloud support	1		3	4	5
Cluster Management Complexity	4		3	3	4
Cluster Scalability	2		4	4	3
Data querying possibilities	3		4	4	5
Database Support	4		5	3	5
Disaster-Discovery	2		1	4	1
Ease of Use	2		3	3	4
High-Availbility	2		1	4	5
Install complexity	2		3	2	3
Licensing and Pricing	2	x	1	5	2
ML support	4	x	5	5	5
Opearting System Support	4	x	2	3	2
Performance	4	x	5	4	4
Programming language support	4	x	4	4	4
Streaming Analytics	1		5	5	5
Support of secondary services	3		3	5	4
Third party module integration	2		2	4	1
User Support & Community	2		2	4	3
Expertise	2		4	5	3
Total Score			182	**209**	198

Legend:	(bad) 1 (is okay) 2 (mediocre) 3 (good) 4 (ideal) 5

Table 3: Weighted Comparison Table of Hadoop Distributions

Table 3 clearly shows that there are almost no significant differences between the selected distributions. Although there are a few deviations such as disaster-discovery or pricing model, the overall distribution of points is relatively the same. From the comparison matrix it can be deduced that Hortonworks HDP achieves the highest score and is probably the best option for our project at the moment. It is possible that after the first test phase (see Chapter 5), it turns out that the distribution is not convenient after all which may require to update the evaluation matrix. Therefore table 3 can be considered as a continuous iterative updateable weighted comparison matrix that will likely be updated on further sprints.

Another representation of the evaluated Hadoop distributions, a spider chart may be appropriate as it makes it easier to read and to identify outliers or similarities even quicker as with a raw table.

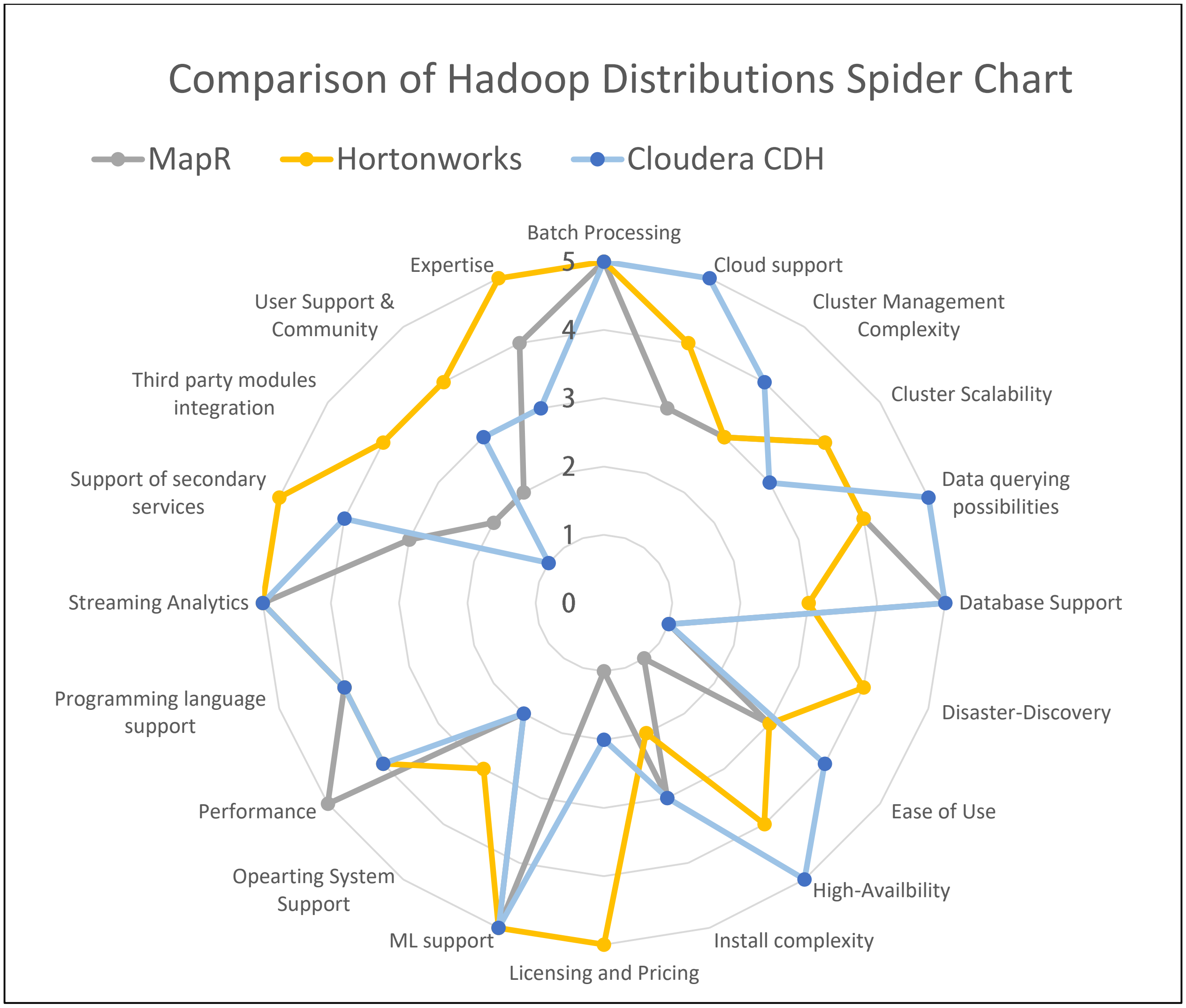


Figure 2: Spider chart of compared Hadoop distributions

The KO criteria from table 3 are all satisfied by the selected Hadoop vendors. One criterion that stands out is the license model, which was rated with 1 point for MapR and 2 points for Cloudera. This is due to the fact that the price models of both vendors are not transparent. So Cloudera offers a free community edition but to use the Cloudera Manager (the real strength of Cloudera) you have to pay again. Only Hortonworks offers a complete Open Source package with except the Hortonworks business support is fee required. However, Hortonworks has a very active community as well.

MapR comes off worst with the criterion “Support of secondary services”, because there is no Spark and HBase support in the "Converged Community Edition" [12], but both are important Hadoop components for our project. So if we wanted to use MapR, we would have to use the paid version. There are also some other differences in figure 2 such as Disaster-Recovery which is only on Hortonworks’s HDP completely free.

An interesting aspect of the comparison is the performance criterion where MapR has the most points. Since this criterion is a KO-criterion that means it is a very important component for our project, it makes sense to examine the performance comparison between the distributions in a more detail fashion.

4. Performance Comparison with micro benchmarks

For a performance comparison it is good to know how fast they are compute when running in a concurrent mode. For this task MapReduce Jobs like WordCount or DFSIO Read/Write might be helpful. Following figures are extracted from a sophisticated evaluation work by Altoros [19].

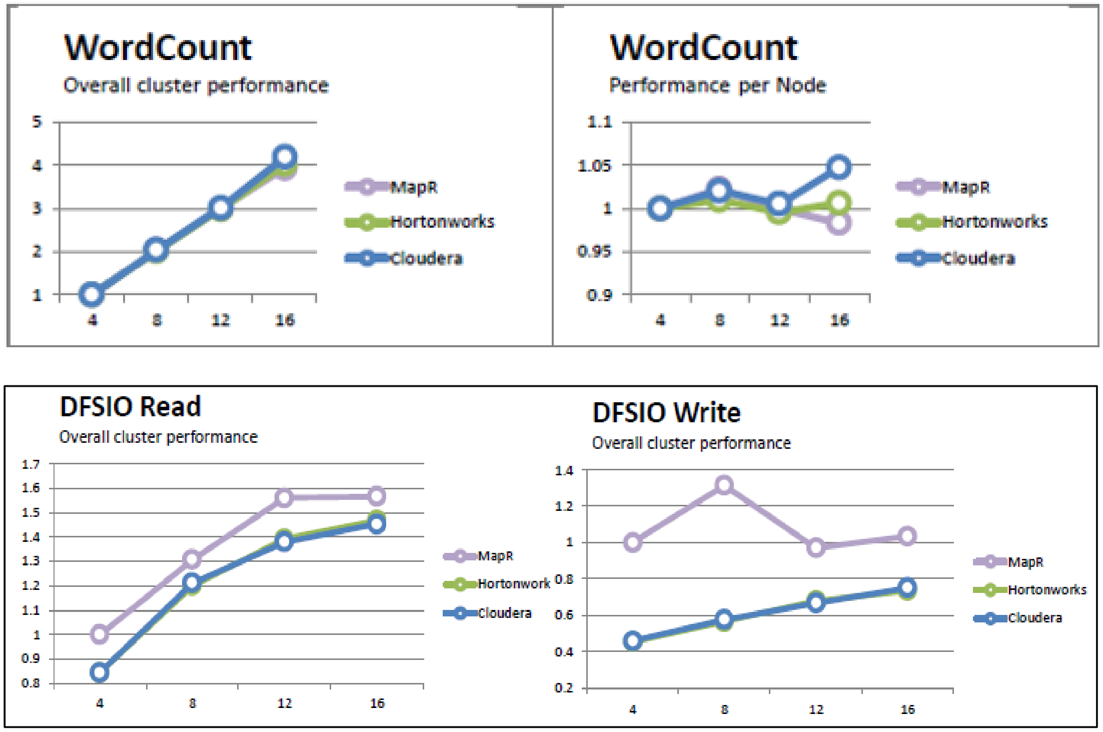


Figure 3: Micro benchmarks on DFSIO / WordCount MapReduce Jobs

For the performance comparison of figure 3 Hadoop cluster each one with 16 nodes have been established in order to measure computation time. Interesting point is that all 3 Hadoop distributions almost achieve the same overall speed on the famous WordCount MapReduce job. But if we look at the performance per node then there is a small difference between them. This could be due to a certain error rate in the execution of the job. If the same test were repeated, the results would be negligibly different. The DFSIO

Read/Write (figure 3) job shows MapR is definitely faster than Hortonworks and Cloudera. The reason for this might be the fact that MapR uses its own HDFS which is according to the vendor seven times faster than the original one from Apache. This is the reason why it gets 5 points in the evaluation matrix (see table 2) whereas other ones only 4 points. Hortonworks and Cloudera seems to have the same performance on the DFSIO job because both using Apache HDFS.

In overall, Hortonworks HDP wins the competition for the moment, because they have the longest experience and are completely compatible with secondary services. Based on the evaluation matrix from chapter 3 and the performance measurements, it can be deduced that MapR is the most powerful Hadoop distribution on the market today, but when considering the other criteria, Hortonwork's Hadoop is simply more convincing. Especially the complete Open Source guarantee at Hortonwork is a decisive criterion for the choice of this distribution.

Beside from the comparison, I would generally recommend to use rather Spark than Hadoop since Spark is around 100 times faster due to in-memory processing. Also Spark provides the most important libraries (ML, Streaming …) for data science and works well on top of Hadoop (thanks YARN). It can even be installed in a standalone manner, even though it doesn't benefit from distributive multimode computing.

5. Installation of sample Hadoop distribution on 4 VM's

Hortonworks offers a configurator on their website that can be used to check products, operating systems, databases, browsers, JDKs and supported processors. It is noticeable that Ubuntu is only supported up to 16.04 LTS. However, our VMs are already v.18.4 LTS. This means Hortonworks does not official support our installed OS [20]. Also Cloudera CDH 5.15 [21] and the newest MapR 6.1 distribution [22] only support Ubuntu 16.04 LTS (Xenial). Anyway, it is still a try worth to set up Hortonworks Hadoop Data Platform (HDP) Cluster on our virtual machines since the vendors are not explicitly warning Ubuntu 18.04 is not supported. So it may work. For the installation of HDP 3.0.1 the Ambari Wizard [23] will be used.

First of all, we have to configure the ***etc/hosts*** since every node in the cluster must be able to communicate with each other. The hosts file should look like this:

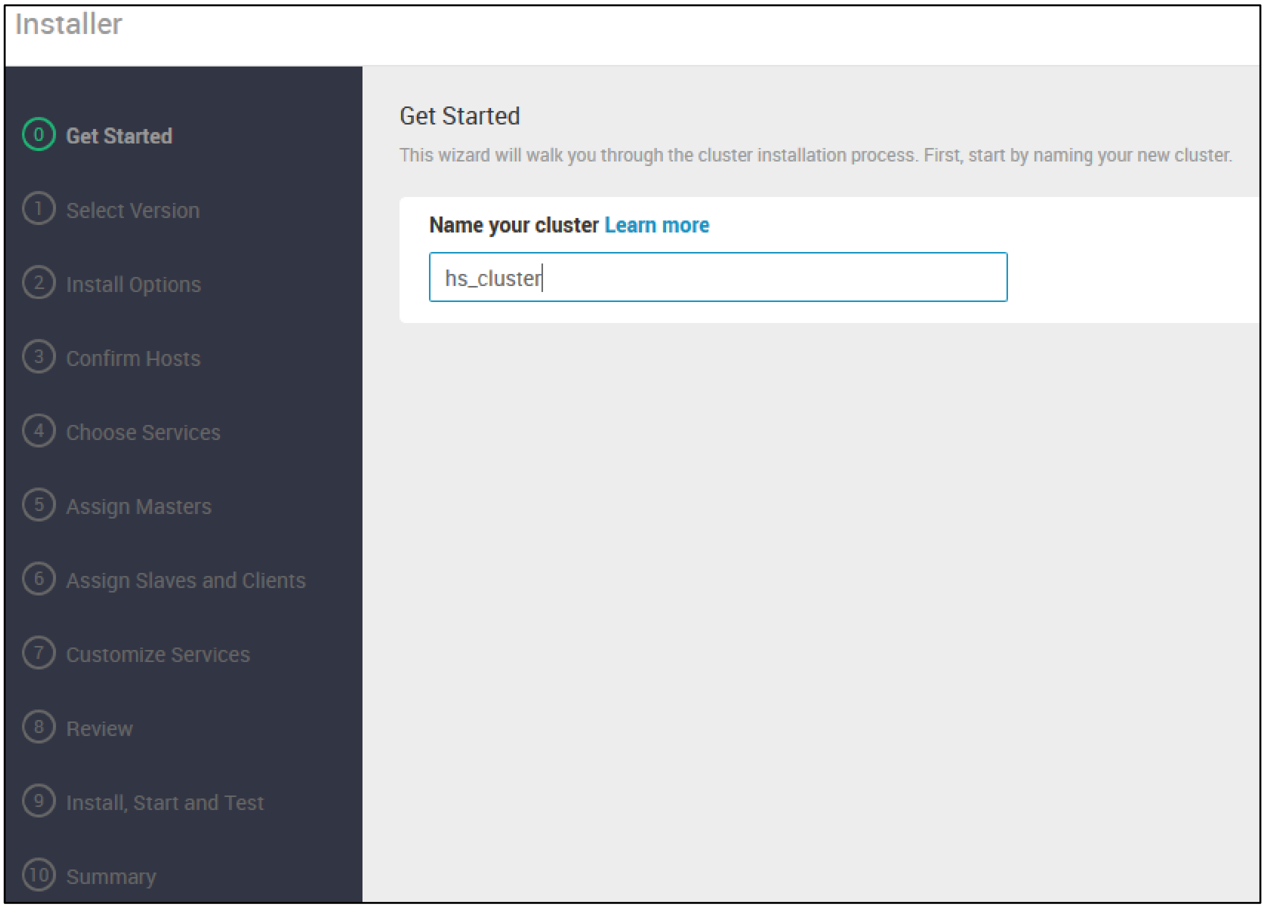
```
127.0.0.1      localhost


141.59.29.111 i-hadoop-01.informatik.hs-ulm.de i-hadoop-01
141.59.29.112 i-hadoop-02.informatik.hs-ulm.de i-hadoop-02
141.59.29.113 i-hadoop-03.informatik.hs-ulm.de i-hadoop-03
141.59.29.114 i-hadoop-04.informatik.hs-ulm.de i-hadoop-04
```

Table 4: etc/hosts configuration file

After that step, it is necessary to add public key authentication with SSH. The master node (**i-hadoop-01**) should login to its worker nodes without using a password. We can achieve this goal by generating a private / public key pair and distribute the public key to the 3 worker nodes by using **ssh-copy-id** command. Also for using Ambari the Hadoop user need sudo execution without password. Adding an additional line to **visudo** configuration file should solve that issue.

At next, all nodes need to have Java installed. Since Ubuntu 18.04 doesn't come up with a default Java JDK we install Oracle JDK 1.8 manually on each node. After that it is time to download the Ambari repository file (http://publicrepo-1.hortonworks.com/ambari/ubuntu16/2.x/updates/2.7.1.0/ambari.list) to a directory on our Hadoop master host. The Ambari v. 2.7.1.0 is used for installation. With the command **apt-get install ambari server** the server will be installed on i-hadoop-01. Afterwards, we can start the server. Now we can go to the Ambari surface via following link: http://i-hadoop-01.informatik.hs-ulm.de:8080. The default login credentials are **admin admin**. After successful login, the real installation process begins.



Get Started

This wizard will walk you through the cluster installation process. First, start by naming your new cluster.

Name your cluster Learn more

hs_cluster

Figure 4: Ambari Wizard Installer

Next, we click through the installation routine until the point appears where the cluster nodes are defined. At this point the fact that we are using Ubuntu 18.04 will cause some troubles. If we adding the hosts Ambari is trying to check some dependencies and is running also some check routines. It will fail to add the worker nodes to the cluster since OS is not supported (termination condition). The solution to this problem would be a downgrade of the OS but we don’t have the privileges to do this. So I manually changed the **/etc/issue**, **etc/lsb-release** and **etc/os-release** file to Ubuntu 16.04 version. Of course, a backup of the original ones has been created as well. After this workaround, the check condition will not fail because Ambari consider our OS as Ubuntu 16.04. Obviously, that’s a dangerous operation, so we change it back to original state after installation has been completed.

In the next step we are choosing our Hadoop services that may be relevant for our data science project. For a start we choose following services: HDFS, YARN, MapReduce2, Tez, Hive, ZooKeeper, Ambari Metrics, SmartSense, Spark2 and Zeppelin Notebook. Interesting to see is that the proprietary service SmartSense is the only one that cannot be deselected. We have to install it, even if we don’t want to use it. That’s certainly not very user-friendly. However, it is simple to add further Apache services on running Ambari but more difficult to uninstall them so that I am not enabling all possible Hadoop services from the beginning. In addition, some services are not for interest at the moment so they would only consume storage without having a practical effect.

After some additional configuration steps we can choose which services should run on master or worker nodes. Also we have to decide which node should be data node and node manager. In this cluster configuration all 4 VMs are data nodes as well as node managers. A database connection is also mandatory at this step. Depending on selected services there are several database connections. For instance, the service Hive needs a working SQL database connection. We can choose between MySQL (MariaDB), PostgreSQL and Oracle. For a first test, MySQL is used as Hive database. We can change the database connection setting on Ambari at any time.

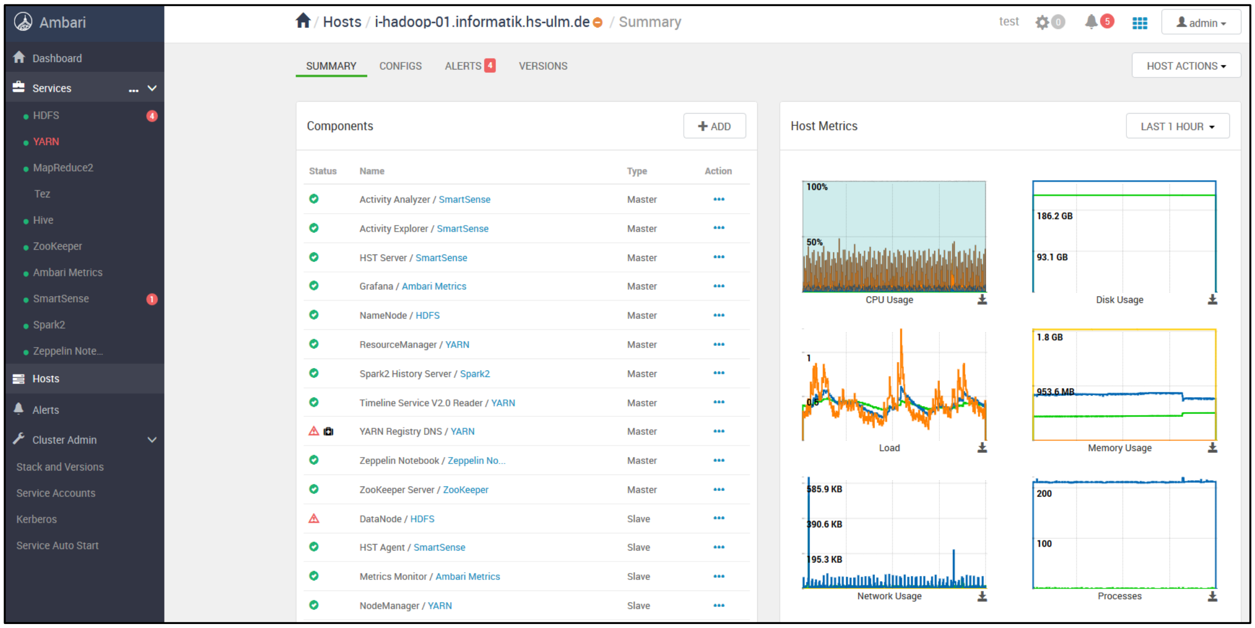
If the configuration has been finished, a summary with all decisions taken is shown. That’s the last possibility to change a fundamental Hadoop setting. It is worth to check all configurations carefully. Afterwards there is no go back option and the uninstall routine of Ambari is complicated and might lead to tremendous work of removing operations. That is also negative criteria of HDP and Ambari.

As soon as the installation completes we can switch to the Ambari monitor. Unfortunately, almost all selected services were offline or could not start properly. A deeper investigation showed that there were some permission problems with HDFS and the Hadoop user. The HDFS folder was only for root user but not for user “Hadoop”. With the command **chown -R hdfs:hadoop /hadoop/hdfs** the correct permissions could be granted.

Another problem was that YARN could not start the name nodes due to connection refused error messages. As **netstat -tupln | grep 50070** shown the service was only running for localhost address. After changing the binding address to **0.0.0.0** on HDFS, the communication between the cluster nodes was working and the name nodes could properly start. However, after fixing this issue, the most services were running. The community of Hortonworks was quite helpful in this case. I got an answer within 4 hours whereas for getting an answer in Cloudera’s community took 3 times longer.

The data node from the master node was not running correctly. It could be started from Ambari and after few seconds the data node went offline. A look at the error log of this node shown that the cluster id is already used by another node. Removing the HDFS data folder of the data node solved this issue because afterwards it generates a new cache file with a new cluster id.

Following screenshot shows a (successful) completed Ambari and HDP installation on our VMs.



Status	Name	Type	Action
✓	Activity Analyzer / SmartSense	Master	•••
✓	Activity Explorer / SmartSense	Master	•••
✓	HST Server / SmartSense	Master	•••
✓	Grafana / Ambari Metrics	Master	•••
✓	NameNode / HDFS	Master	•••
✓	ResourceManager / YARN	Master	•••
✓	Spark2 History Server / Spark2	Master	•••
✓	Timeline Service V2.0 Reader / YARN	Master	•••
⚠	YARN Registry DNS / YARN	Master	•••
✓	Zeppelin Notebook / Zeppelin No...	Master	•••
✓	ZooKeeper Server / ZooKeeper	Master	•••
⚠	DataNode / HDFS	Slave	•••
✓	HST Agent / SmartSense	Slave	•••
✓	Metrics Monitor / Ambari Metrics	Slave	•••
✓	NodeManager / YARN	Slave	•••

Figure 5: Ambari Surface, Host metrics on master node

Ambari has been set up and most Hadoop services are running (see Figure 4). Ambari shows lot of statistics that might be helpful for cluster management. We can control each service and either turn it off or set it into a maintenance mode. The ladder one encapsulates the service without influencing other services (high availability). Overall, Ambari is a powerful Hadoop cluster management tool that has lot of control options and supervising tools but also lacks in removing services and usability (e.g. no Ubuntu 18.04 support, only Python 2.7.x support…). The fact, that there is no official uninstaller from Hortonworks makes HDP with Ambari a risky installation. On the other hand, it is completely open source and has a strong community. Furthermore, HDP 3.0.1 comes up with a bunch of useful Hadoop services and well documentation so that we will likely stay on HDP. Now it is time to run some mapreduce jobs on our newly created Hadoop cluster.

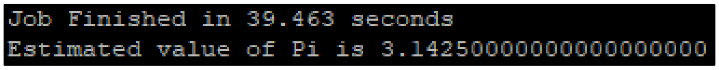
6. Run MapReduce Jobs

With 4 workers (master is also a worker node) a significant speed advantage over single computing should be achieved.

First, with a MapReduce job, the accuracy of the decimal places of PI is determined using the quasi Monte Carlo method. This MapReduce job is also offered by the Apache foundation (Source). In the console of the master node the MapReduce job can be started with "yet another resource negotiator (YARN)" as follows:

```
yarn  jar  /usr/local/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.1.1.jar pi 16 1000
```

The PI job calculates 16 maps where each map process contains 1000 samples. After a short time, the job is finished and the following output is achieved:



```
Job Finished in 39.463 seconds
Estimated value of Pi is 3.14250000000000000000
```

Figure 6: PI Quasi Monte Carlo method executed as Cluster job

The same test has been tested with 2 worker nodes (i.e. 2 of the 4 workers were temporarily disabled via Ambari). After that the job took much longer (about 56 seconds). This shows that the Hadoop cluster works and already has a performance advantage over single computing.

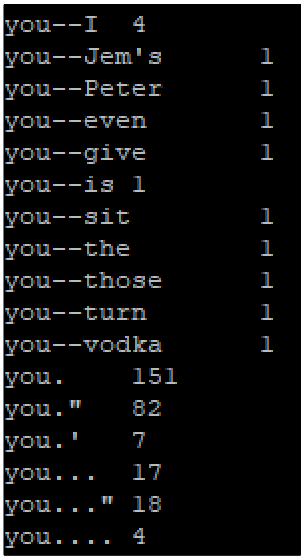
The next test was a simple WordCount job in Java that calculates the frequency of unique words in an input file distributed across the cluster. A MapReduce job consists of at least a mapper method, a reducer method and a driver method which is often the start method. Before executing the new job one has to put the file of interest on Hadoop’s HDFS. This is simply done as shown below:

```
hadoop fs -put big.txt
```

The put command will upload the file into the HDFS (i.e. it is stored fault-tolerant and replicated 3 times). After that, the following command runs a WordCount job on this txt file:

```
hadoop jar hadoop-mapreduce-examples-3.1.1.jar wordcount wordcountFile big.txt
```

Note that now “Hadoop” is used, which starts also the YARN service. There is no difference between executing this with the Hadoop or YARN command. (Although latter one is newer and should be used). The output of the “reduced” file looks like figure 7.



```
you--I  4
you--Jem's      1
you--Peter      1
you--even       1
you--give       1
you--is 1
you--sit        1
you--the        1
you--those      1
you--turn       1
you--vodka      1
you.    151
you."   82
you.'   7
you...  17
you..." 18
you.... 4
```

Figure 7: Extract from the output of the WordCount job

Quick to recognize that the words were separated and counted by spaces. In fact, the input text was split afterwards and each word was mapped as "key value pair", where the key in this case is the word. The original text file was 6.4 MB, while the newly created file is only 0.93 MB in size. This shows another potential benefit of the cluster, namely the possible separation of files to minimize the total size of the initial data set. The running time was around 10 seconds. The same job has been executed locally on a commodity Windows client in NetBeans as well and it took around 8 minutes.

There is a plenty of further typical MapReduce jobs (e.g. terasort, randomwriter, sudoko…), but for an evaluation these two jobs should already show that the cluster is working and apparently faster than a single node.

7. HDP Issues

Due to the workaround in Chapter 5, a few services could not start properly. Also, permissions for some subfolders were set incorrectly, so that permission denied errors occurred when running the services. During the test phase some other problems were discovered, which are described below. Table 4 also shows the solution that was found to resolve these issues.

Issue	Solution
No starting NameNodes, Connection refused.	Go to HDFS -> Filter at https 0.0.0.0 and localhost address on master node http 127.0.0.0:50070 --> 0.0.0.0:50070

Issue	Solution
	Hive could not start on Node 2 since database was missing (mysql) Permission on HDFS was set to root and not to user hadoop

Wrong Permissions on HDFS	`tail hadoop-hdfs-namenode`
	`chown -R hdfs:hadoop /hadoop/hdfs`

Database is not set on the Ambari Server	`ls /usr/share/java/mysql.jar`
	`ambari-server setup --jdbc-db=mysql –jdbc-driver=/usr/share/java/mysql.jar`

Hive Database not found on i-hadoop-02	• Start mysql • Show databases; • create database hive; • CREATE USER 'hive'@'localhost' IDENTIFIED BY 'hive'; • GRANT ALL PRIVILEGES ON . TO 'hive'@'localhost'; • CREATE USER 'hive'@'%' IDENTIFIED BY 'hive'; • GRANT ALL PRIVILEGES ON . TO 'hive'@'%'; • GRANT ALL PRIVILEGES ON . TO 'hive'@'localhost' WITH GRANT OPTION; • GRANT ALL PRIVILEGES ON . TO 'hive'@'%' WITH GRANT OPTION; • FLUSH PRIVILEGES;
Bind address in mysql-conf (0.0.0.0):	• netstat -tupln | grep 3306 • Nano /etc/mysql/mysql.conf • Change bind-address to 0.0.0.0 • Systemctl mysql restart • Service mysql restart
YARN DNS Server not starting, connection refused	• Change Port 53 to other Port • Set right permissions on YARN folder
Master Node worker is not starting (Conflicting cluster id)	• Stop Ambari Server • Delete data directory of data node in hdfs (do backup of folder) • Create new empty one • sudo chmod 750 data • Restart Ambari Server
Zeppelin Notebook Python standalone support not available	• Stop service • sudo wget archive.apache.org/dist/zeppelin/zeppelin-0.8.0/zeppelin-0.8.0-bin-all.tgz • sudo tar -xf zeppelin-0.8.0-bin-all.tgz • Move interpreter to /usr/hdp/current/zeppelin-server/interpreter/ • sudo chown -R zeppelin:zeppelin /usr/hdp/current/zeppelin-server/interpreter/ • Start service
Umask is changing from 0022 to 0002	• sudo gedit ~/.bashrc • Add line umask 022

Issue	Solution
	- Logout and login
FAILED [2018-12-08 13:36:00.609]java.io.IOException: mkdir of /hadoop/yarn/local/filecache/60_tmp failed	- YARN permissions - sudo chown yarn yarn - sudo chmod 750 yarn

Table 4: Problems and solutions of HDP

The described problems and their solutions from table 4 show that many setting options should be checked carefully during the Ambari installation routine (e.g. if a MySQL database exists). Furthermore, it can be assumed that most of these issues can be explained by the "officially" unsupported operating system version. However, after the fix, all Hadoop services in Ambari had a green status point.

8. Recommendation

The evaluation showed that HortonWorks with the Hadoop distribution "HDP" is best suited for our goals. Ambari provides an efficient, comprehensive cluster management tool for monitoring and controlling the Hadoop cluster. In addition, an active community and complete open source freedom can convince more than its competitors. Although all 3 Hadoop distributions may have their individual advantages and disadvantages (see evaluation matrix, table 3), HortonWorks was able to convince us due to the reasonably good documentation and the easy to follow Ambari installation routine. It should be noted, however, that the Cloudera Manager installation wizard also has good documentation and a comparable installation effort. However, I liked the Ambari interface more than the CDH one. In addition, Cloudera's pricing is not entirely transparent and third-party module integration is rather difficult. For example, since CDH 5.11 there has been a proprietary "data science workbench" service replacing Jupyter and IPython notebooks. HortonWorks, on the other hand, offers Zeppelin notebooks that resemble the famous Jupyter notebooks. Apart from that, Jupyter could also be installed as a third module during the test phase of HDP, even though such installations are naturally risky and are not officially supported by HortonWorks.

It is interesting that MapR, HortonWorks and Cloudera offer Docker images for immediate testing. For example, MapR offers a VMware image free of charge. However, all these packages have the disadvantage that they only have a single node cluster and limited functionality. For example, I could not select any services or create new ones in MapR and the function to add more worker nodes to the existing cluster was also not available. Therefore, such images or sandboxes are ideal for a quick check of the user interface and some standard functions, but rather insufficient for a complete range of functions or a productive system.

In conclusion it can be said that all 3 Hadoop distributions would meet the requirements of a Big Data project or our project perfectly, but HortonWorks could simply convince more with the overall score than the more commercially oriented counterparts. Ultimately, the performance also depends on the hardware of the individual cluster nodes. Due to a low physical RAM, MapReduce jobs even with MapR's high performance HDFS would be processed slowly by YARN and would probably lead to MemoryoutofExceptions. Because Hadoop and every single service on top needs resources and these can quickly bring a commodity computer with standard equipment to its knees. In any case, the HDP cluster is ready for big data tasks and will be extremely helpful to us in the further course of the project. With Apache Spark, even better performance values might be achieved due to its "in-memory" engine. The test cluster with the 4 VMs also showed which special features and requirements you should pay attention to (e.g. operating system version).

9. Data Profiling

Data profiling is the process of reviewing source data, understanding structure, content and interrelationships, and identifying potential for data projects. For the project, the Santander Bicycle data will be profiled more closely [24]. For this I mainly used Zeppelin notebooks and the HDP cluster with 4 worker nodes initialized in chapter 5. Zeppelin works similar to Jupyter notebooks and also supports magic commands. However, Zeppelin stores the notebooks in .json format, while Jupyter Notebooks (Python) uses .ipynb. A corresponding import of Zeppelin notebooks into Jupyter is therefore not possible and should be considered when working with one of them. Furthermore, as already mentioned, HDP only supports Python 2.7.x, so all Python code in Zeppelin is Python 2.7.x as well. (The Nominatim server and Django already use Python 3.7.x).

Getting and prepare the data

The Santander bicycle data is available on the public Transport for London TfL website.
The structure of these data is as followed:

- **CycleCounters:** has information about speed, bike details (length, wheels, axles, class…), direction, timestamps and serial number.
  - Blackfriars May and June and July
  - Embankment May and June and July
- **CycleParking:** contains proprietary map material
- **CycleRoutes:** contains geographical spatial data (GeoJSON) and .KML files
- **Usage-stats:** information about rental stations, duration, start / end time

With regard to the folder structure shown above, **CycleRoutes** and **Usage-stats** are decisive. In "CycleCounters", apart from the maximum driven speed (56 mph!) with a borrowed Santander Cycle, no noteworthy information is contained. Therefore, only the CycleRoutes and Usage-stats data are considered in the first step of Data Profiling. The CycleParking folder could contain useful map material, but I only know from Google Earth that can read .kml files and that it is neither compatible with Geopandas nor with Nominatim. Most raw data are available either as **.csv** files or as **.xlsx** files.

One problem that arose when reading the usage-stats was that the folder contained 156 csv sheets and each file was about 30 MB in size. Reading and concatenating as Pandas DataFrame unfortunately completely occupied the maximum free memory of the master node, so the Python job could not be finished and was in an endless run mode. To work around the problem, I started a Spark job with the PySpark API on the cluster instead of the Panda DataFrame. The spark job had created a DataFrame from 156 raw files within 12 seconds (i.e. it read 38.122.372 lines of raw text!). What we get here is a so called Resilient Distributed Dataset (RDD) or Spark DataFrame which, in contrast to the Pandas DataFrame, is redundant and distributed over the Cluster DataNodes. With the following PySpark method an RDD can be generated:

```
%spark2.pyspark

rdd_usage = spark.read.csv("/user/hadoop/usage-stats/*.csv", header=True)
```

Of course the files of the folder “usage-stats” need to be stored on HDFS before it can be read in with Spark. There are several ways of doing this, we used just a Zeppelin cell with magic command shell (%sh) to execute shell based commands within the notebook. With that one can `execute hdfs dfs -put /home/hadoop/TFL_Cycling_Data_raw/usage-stats  /user/hadoop/usage-stats` to push the files on the datanodes of HDP. Remember the default replication size of Hadoop is 3, i.e. the files are replicated on 3 worker nodes but not on 4, which should still be resilient enough.

The next decisive step is to prepare the usage-stats data and clean it up a bit. If we read in the data as described above, our Spark DataFrame has the following columns:

```
|Rental Id|Duration|Bike Id| End Date|EndStation Id| EndStation Name|
Start Date|StartStation Id| StartStation Name|
```

Since we only want to plot the bicycle stations at this time, the Rental ID columns and the date columns have been removed. This has the advantage that the DataFrame is already smaller and we don't carry any unnecessary ballast. Furthermore, NaN values have been replaced by mandatory "0" and duplicates have been removed. The TFL also offers numerous .xml based live feeds, from which we can get the coordinates (longitude and latitude) to the rental stations. We can also use this to fetch the coordinates:

```
%spark2.pyspark
import requests
from xml.etree import ElementTree as ET
import pandas as pd

#Getting the coordinates tuples of the rental stations
site = "https://tfl.gov.uk/tfl/syndication/feeds/cycle-hire/livecyclehireupdates.xml"
response = requests.get(site)
root = ET.fromstring(response.content)

#fetch relevant information
id_list = [int(root[i][0].text) for i in range(0, len(root))]
name_list = [root[i][1].text for i in range(0, len(root))]
lat_list = [float(root[i][3].text) for i in range(0, len(root))]
lon_list = [float(root[i][4].text) for i in range(0, len(root))]
capacity_list = [int(root[i][12].text) for i in range(0, len(root))]

#zip tuples
all_locs = pd.DataFrame(list(zip(name_list, id_list, lat_list,
                                 lon_list, capacity_list)), columns = ["name","id","lat","lon","capacity"])

#zip
all_locs.to_csv('/home/hadoop/TFL_Cycling_Data_raw/rental_stations_saved.csv', header=True, index=None)
print(all_locs.shape)

#786 unique rental stations
all_locs.head()
```

This gives us not only the longitude and latitude, but also the maximum bicycle capacity of any station in London. This information could still be quite useful. As you can see, we have 786 unique bicycle stations in London (which are quite a lot).

Remark: Livecyclehireupdates.xml contains also following attributes:

- installed
- locked
- installDate
- nbBikes
- nbEmptyDocks
- nbDocks

Given that there are 786 stations across London (at least at the time of writing), this allows for 786 * 785 = 617.010 possible journey combinations if we ignore those that start and end at the same station. The next step is some data cleansing steps such as sorting, renaming and converting data types. PySpark offers similar operations for the RDDs as for Pandas DataFrames. However, some pandas such as "iloc()" cannot be transferred to Spark RDDs. Therefore Spark offers the option to directly execute SQL queries in DataFrames. Anyway, the fetched data from the live feed must still be merged with the rdd_usage Spark DataFrame. For this purpose we first saved the Pandas DataFrame locally and then uploaded it to the HDFS. This checkpoint is then read in again with PySpark as RDD. Merging works similar to SQL or/and Python. To do this, simply use the "join" command to link the two DataFrames:

```
%spark2.pyspark
from pyspark.sql.functions import *

##Merging
#Merge cycle_usage RDD with RDD from livecyclehireupdates for
StartStations
ta = unq_rentals.alias('ta')
tb = rdd_rental_locs.alias('tb')
unq_rentals = ta.join(tb, ta["StartStation Id"] == tb.id)

```

In this case we also used alias (similar to the alias of SQL) in order to shorten the joining function name. The rest of the merging stuff is automatically done by the Spark API. After all the data munging steps we got following schema of our combined and prepared usage-stats table:

```
StartStation Id|EndStation Id|Duration|StartStation Address|StartStation
latitude|StartStation longitude|StartStation capacity| EndStation
Address|EndStation latitude|EndStation longitude|EndStation capacity|
```

One can note, that there are also the columns “StartStation ID Used” and “EndStation ID Used” which stands for frequency of rented bikes on a single station. This gives us an indication of how popular the station is in comparison with other ones. It might be also used as a weighting factor for plotting on maps which will be described in just a moment.

With this, we can begin to plot statistics. But at first let compare the size of the new file with the total size of all loaded cycle-usages. The total size was around 5.4 GB (which could not be simply loaded into a Pandas DataFrame) whereas the new DataFrame has only 0.9 GB size. The reduction of the size can be explained by the fact that we removed some unused columns and dropped duplicates. This means also we are now able to use Pandas because to load 1 GB into a local memory should be possible for most common computers. In fact, the Hadoop test environment has shown its functionality and legitimation for the using in big data projects.

Plotting the data

Let’s see if we can compare the frequency of used rental stations between StartStation and EndStation. We can not use swarm plots or any other kind of plots with fine granular details since “matplotlib” or Python would try to draw over 6 million dots which will easily exceed the local RAM of the master node. Beside from that, we also would see to many points at the same location since lot of stations are drawn several times. A better plot in my opinion is to use a box plot as following figure shows:

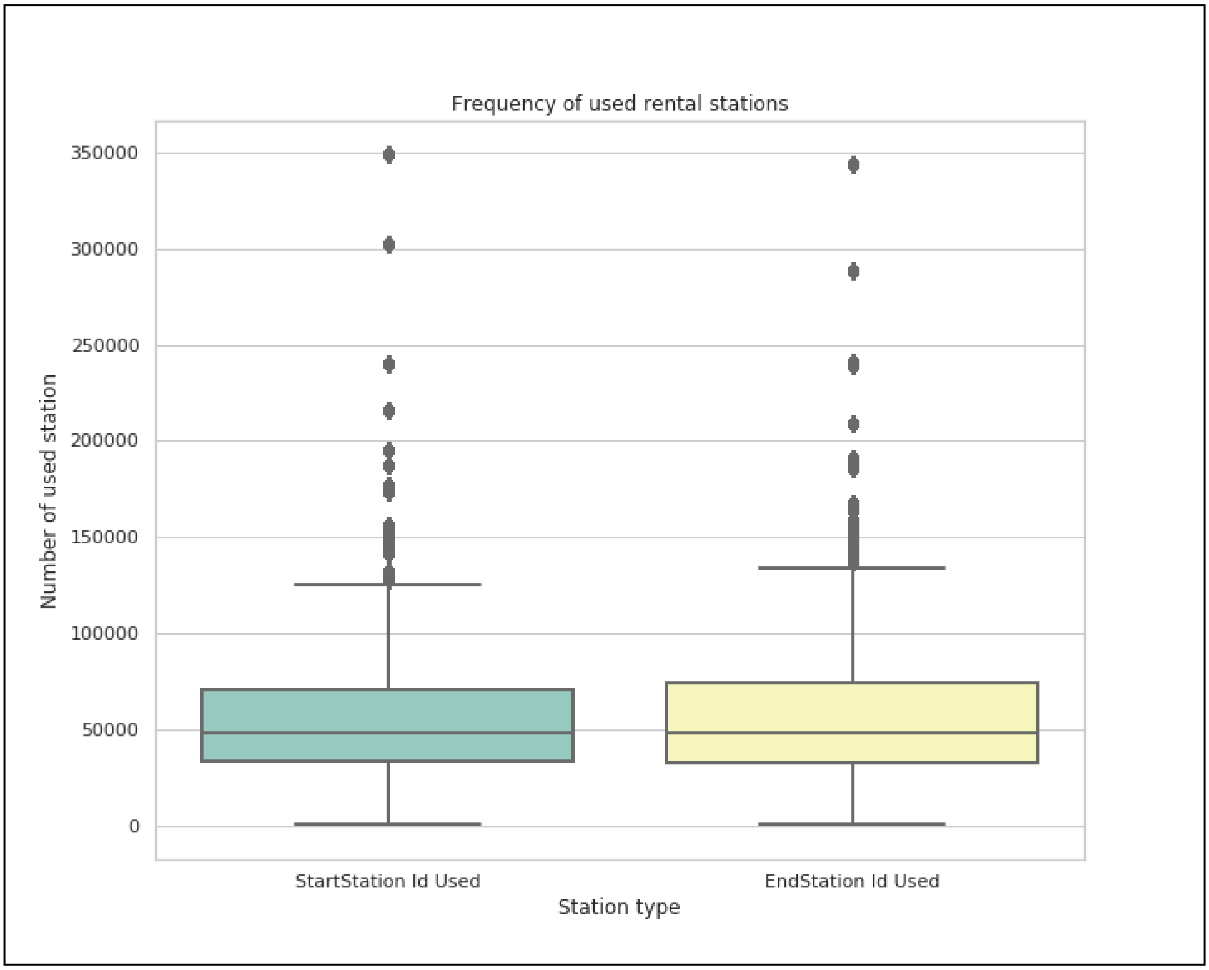


Figure 8: Frequency of used StartStations and EndStations

Wie wir sehen aus Abbildung 8, überwiegt die Nutzung von Endstationen geringfügig den StartStationen, was lediglich bedeutet, dass mehr unterschiedliche Endstationen als unterschiedliche Startstationen genutzt wurden. Scheinbar gibt es ein paar sehr beliebte rental stations wie die Ausreißer aus Abbildung 8 zeigen. Es ist noch erwähnenswert, dass die Daten alle von 2016 – 2018 erfasst wurden. (Die Jahr 2014-2016 sind in diesem Fall noch nicht berücksichtigt).

The next plot should serve us to show the density center of the rental stations locations. For this a joint kernel density estimation plot could be helpful. We are plotting longitude against latitude of each single StartStation. The illustrations looks then like this:

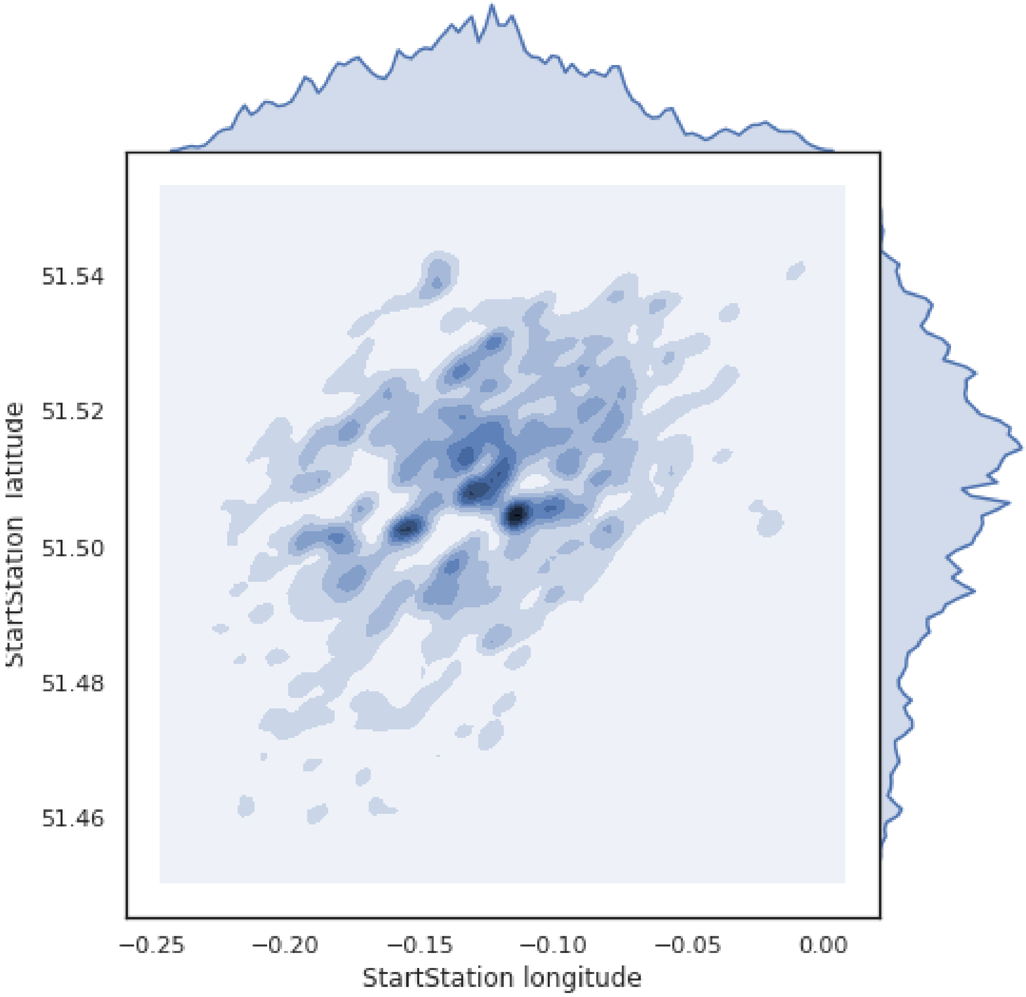


Figure 9: Frequency of used StartStations and EndStations

The centroid or center of this plot is indicated by the dark spots. There are the most rental stations as also the frequency distributions (i.e. histograms) on the right border and on the top are clearly showing. Apart from that center of gravity, on the right corner there are also rental stations placed (guessing between longitude: -0,01 and latitude 51,6) which indicates that these rental stations must be placed in an outer borough of London.

At a next, it would be useful to show the rental stations on a map based on their usage. As already mentioned we have 786 unique rental stations so that we can drop the duplicates for this case. In fact, we have saved the number of used stations on a separate column. At this step we prefer to use Jupyter on a local machine since Zeppelin on the cluster is not supporting IPython Display which is required by third-party packages such as folium. So we loaded the prepared cycle-usage data into a local Pandas dataframe. Now we can plot a heatmap for the rental stations:

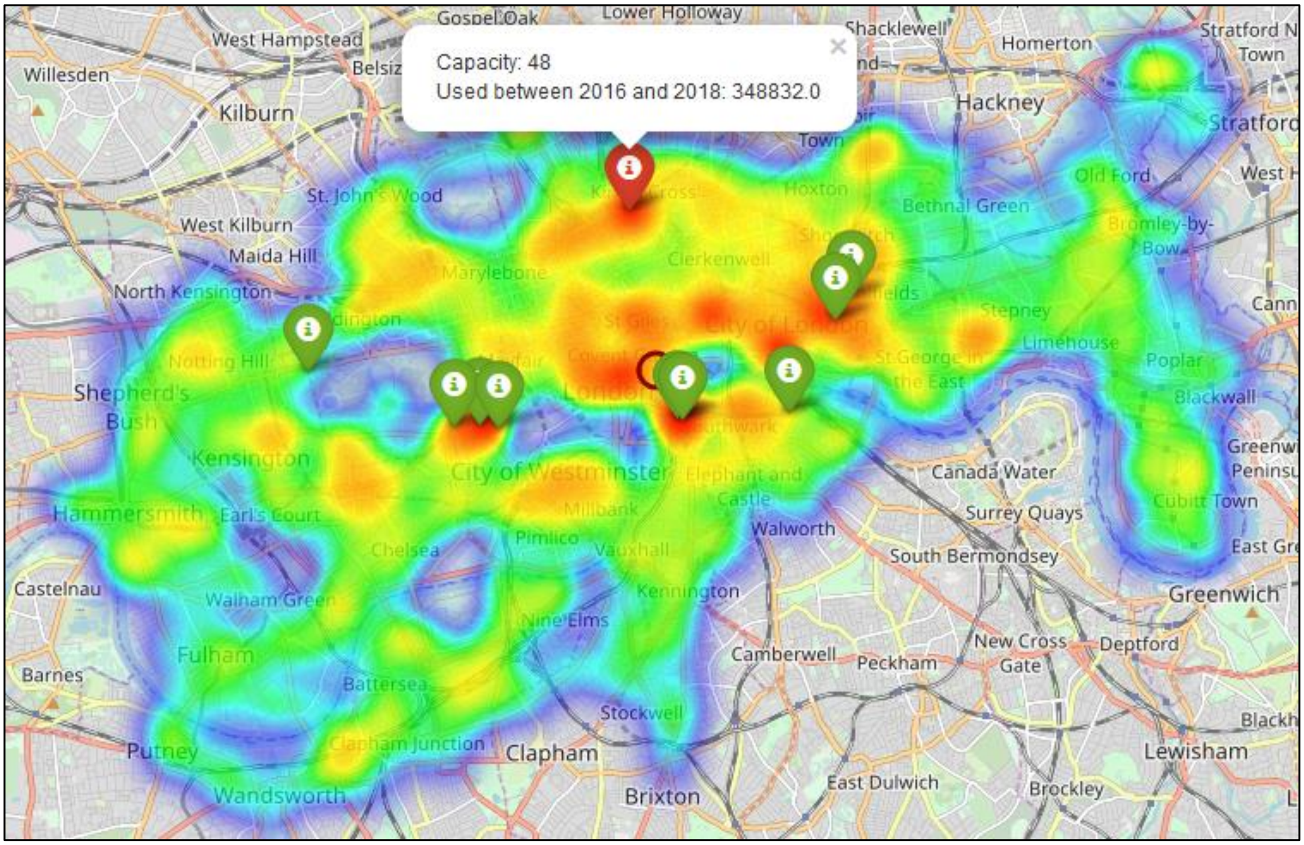


Figure 10: Top 10 used Santander rental stations

The red marker in figure 10 indicates the most often used bicycle station in London in last two years. This is not surprisingly because close to the biking station there is London train station. Probably many travelers who are arriving by train are using a Santander cycle. However, this interactive map has been saved and uploaded on a webserver on the Hadoop cluster (link: http://i-hadoop-01.informatik.hs-ulm.de/most_stations.html)

As we can see, there are many rental stations close together. For a first impression this should be enough as one can already see the capacity of each single Santander rental bike station by clicking on the marker and also the location of each station is visible on the map. As an addition one can see the frequency of each station by looking at the sizes of the heat spots or/and reading marker details.

Conclusion of the file sizing problem

1. The sizing problem can be well handled by Spark
2. The HDP is working and storing the cycle-usage files on HDFS

3. For plotting tasks packages like folium or seaborn cannot be used with Spark DataFrame
    - Converting back to Pandas DF might exceed local RAM
    - Using Jupyter notebook on local computer for plotting maps (e.g. folium)

CyclingRoutes

The Cycling routes from the folder of TFL [24] are geojson files and represent real geographical routes within London inner city.

There are 5 geojson files in this folder available.

- Central_London_Grid.json
    - Central London Grid: is a set of safer, connected routes for cyclists across central London.
- Mini_Hollands.json
    - Mini-Hollands have features that make cycling feel safer and more convenient.
- Quietways.json
    - Quietways are signposted cycle routes, run on quieter back streets to provide for those cyclists who want to travel at a more relaxed pace.
- super-highways-quietways.json
    - Combination of super highways and quietways
- Cycle_Superhighways.json
    - Cycle_Superhighways are on main roads and often segregate cyclists from other traffic.

The routes should be concatenated by a single GeoPandas GeoDataFrame. Doing so requires to read the geojson files with following scripts:

```
%spark2.pyspark
import pandas as pd
import geopandas as gpd

pd.set_option('display.expand_frame_repr', True)
pd.options.display.max_columns = None

path ='/home/hadoop/TFL_Cycling_Data_raw/CycleRoutes/'
grid_data = ["Central_London_Grid.json",
             "Mini_Hollands.json",
             "Quietways.json",
             "super-highways-quietways.json",
             "Cycle_Superhighways.json"]


cycle_routes_df = gpd.GeoDataFrame(columns=["OBJECTID", "Label",
"Route_Name", "Status", "Shape_Length", "Tag", "geometry", "gridType"])

for i in grid_data:
    print "Load " + i
    df = gpd.read_file(path+grid_data[0])
    #Extract and load json as gridType
    df["gridType"] = str(grid_data[0])[:grid_data[0].index(".json")]
    #Concatenate
    cycle_routes_df = pd.concat([cycle_routes_df, df])


cycle_routes_df.dropna()

#Save it back as shape
cycle_routes_df.to_file(path+"CycleRoutes_Cleaned")

cycle_routes_df.tail()
```

So we get in only GeoPandas DataFrame which contains the 5 Cycling routes. In addition, the type of each route was stored in the column "gridType", so that it can be filtered later for certain routes. For a first plot a Choropleth should show the different routes.

Conclusion

1. The data lacks in documentation and quality (Data Preperation & Cleansing necessary).
2. Some column names can be misunderstood (e.g. SITE_NUMBER <> SITE_ID).
3. Data like SPEED or SPEED_MHP should be stored in one way.
4. Many columns like the axles are not meaningful and useful for our project. (e.g. Cyclecounters has 107 columns!)
5. Raw data seems not to be complete (e.g. only May, June, July)
6. `usage-stats` is very big. Necessity to use all usage-stats?

Outlook for next Sprint

Create similar plots as shown above, but this time with duplicates and times in mind. Because it is quite often the case that between two same bicycle stations several same entries exist at different times with different rental periods. For example, a use case would be to display the day/night change on the map, i.e. do more cyclists ride quietways or super-highways at night between rental stations?

Bibliography

[1] B. Marr, "Big Data: Who Are The Best Hadoop Vendors In 2017?

Available on: https://www.linkedin.com/pulse/big-data-who-best-hadoop-vendors-2017-bernard-marr," ed, 2017.

[2] M. Gualtieri, N. Yuhanna, H. Kisker, and D. Murphy, "The Forrester Wave™: Big Data Hadoop Solutions, Q1 2014," ed: Forrester, 2014.

[3] G. Kirill, "Comparing the top Hadoop distributions

Available on: https://www.networkworld.com/article/2369327/software/comparing-the-top-hadoop-distributions.html," ed, 2014.

[4] Cloudera, "Cloudera - Flexible pricing and licensing options

Available on: https://www.cloudera.com/products/pricing.html," ed, 2018.

[5] M. Heo, "Hortonworks vs. Cloudera vs. MapR

Available at: http://mungeol-heo.blogspot.com/2015/01/hortonworks-vs-cloudera-vs-mapr.html," ed, 2014.

[6] A. Foundation, "Hadoop

Available on: https://hadoop.apache.org/," ed, 2018.

[7] Cloudera, "CDH Components

Available on: https://www.cloudera.com/products/open-source/apache-hadoop/key-cdh-components.html," ed, 2018.

[8] Cloudera, "Installing CDH 5 Components

Available on: https://www.cloudera.com/documentation/enterprise/5-13-x/topics/cdh_ig_cdh5_comp_install.html," ed, 2018.

[9] D. Kumar, "Since Cloudera and Hortonworks are 100% open source, can I use them freely as I would a Linux distribution?

Available on: https://www.quora.com/Since-Cloudera-and-Hortonworks-are-100-open-source-can-I-use-them-freely-as-I-would-a-Linux-distrubution," ed, 2017.

[10] Hortonworks, "Grundlagen der HDP-Sandbox

Available on: https://de.hortonworks.com/tutorial/learning-the-ropes-of-the-hortonworks-sandbox/," ed, 2018.

[11] Hortonworks, "Starting HDP Services

Available on:

https://docs.hortonworks.com/HDPDocuments/HDP2/HDP-2.6.5/bk_reference/content/starting_hdp_services.html," ed, 2018.

[12] MapR, "MapR Editions.

Available on: https://mapr.com/products/mapr-distribution-editions/," ed, 2018.

[13] MapR, "MapR 6.1 Documentation

Available on: https://mapr.com/docs/home/c_ecosystem_intro.html," ed, 2018.

[14] M. Adrian, "IBM Ends Hadoop Distribution, Hortonworks Expands Hybrid Open Source

Available on: https://blogs.gartner.com/merv-adrian/2017/06/21/ibm-ends-hadoop-distribution-hortonworks-expands-hybrid-open-source/," ed, 2017.

[15] Pivotal, "Overview of Apache Stack and Pivotal Components

Available on: http://pivotalhd-210.docs.pivotal.io/doc/1110/OverviewofApacheStackandPivotalComponents.html," ed, 2018.

[16] I. K. Center, "IBM BigInsights

Available on: https://www.ibm.com/support/knowledgecenter/en/SSPT3X_4.2.0/com.ibm.swg.im.infosphere.biginsights.product.doc/doc/bi_features_architecture.html," ed, 2018.

[17] IBM, "IBM Hadoop Testversions

Available on: https://www.ibm.com/analytics/de/de/technology/hadoop/hadoop-trials.html#start-trial," ed, 2018.

[18] Microsoft, "Manage Hadoop clusters in HDInsight by using the Azure portal," *Available on: https://docs.microsoft.com/en-us/azure/hdinsight/hdinsight-administer-use-portal-linux,* 2018.

[19] Altoros, "Hadoop Distributions: Evaluating Cloudera, Hortonworks, and MapR in Micro-benchmarks and Real-world Applications," pp. 1 - 65, 2013.

[20] Hortonworks, "Supportmatrix Hortonworks

Available on: https://supportmatrix.hortonworks.com/," ed, 2018.

[21] Cloudera, "Operating System Requirements

Available on: https://www.cloudera.com/documentation/enterprise/6/release-notes/topics/rg_os_requirements.html," ed, 2018.

[22] MapR, "Operating System Support Matrix (MapR 6.x)

Available on: https://mapr.com/docs/61/InteropMatrix/r_os_matrix_6.x.html," ed, 2018.

[23] Hortonworks, "Apache Ambari Installation

Available on: https://docs.hortonworks.com/HDPDocuments/Ambari-2.7.1.0/bk_ambari-installation/content/ch_Getting_Ready.html," ed, 2018.

[24] TFL, "Cycling Data for Transport for London

Available on: https://cycling.data.tfl.gov.uk/," ed, 2019.