

# DATA INFRASTRUCTURE FOR INTELLIGENT TRANSPORTATION SYSTEMS

# 5

**Andre Luckow and Ken Kennedy**

*IT Innovation and Research, Information Management Americas, BMW Group, Greenville, SC, United States*

## 5.1 INTRODUCTION

Increasing amounts of data are produced and processed using connected transportation systems comprising myriads of sensors deployed in connected vehicles, roadway/roadside equipment, traffic signals, and mobile devices—the Internet of Things. The ability to efficiently collect, process, and analyze this data and to extract insight and knowledge that drive Intelligent Transport Systems (ITS) is critical. The aim of this chapter is to give an overview of infrastructures to support the requirements of CTS applications. To deal with the complex requirements of Connected Transport Systems (CTS), a data infrastructure capable of storing/processing large volumes using different abstractions and runtime systems is required. The Hadoop ecosystem consists of a manifold set of high-level abstractions, data processing engines, for example, for SQL and streaming data, and scalable machine learning libraries. In this chapter, we give an overview of the Hadoop ecosystem and develop a reference architecture for implementing an infrastructure for CTS applications as defined by the Connected Transport System Reference Implementation Architecture [1]. As the sophistication and scale of CTS increase, the need for a scalable infrastructure becomes even more important as the backend IT infrastructure must be able to facilitate the interactions between vehicles as well as the development and deployment of machine learning algorithms for personalization and optimization of the system.

Hadoop [2] is a scalable platform for compute and storage and has been adopted as the default standard for Big Data processing at Internet companies and in the scientific community. A rich ecosystem has emerged around Hadoop, comprising tools for parallel, in-memory, and stream processing (most notably MapReduce and Spark), SQL and NoSQL engines (Hive, HBase), and machine learning (Mahout, MLlib).

The aim of this chapter is to develop an understanding of CTS applications, their characteristics, and requirements with respect to data infrastructure. In Section 5.2 we analyze CTS applications and their characteristics. We map these requirements to a confined technical architecture for a data infrastructure in Section 5.3. We describe the high-level infrastructure in Section 5.4, and the low-level infrastructure in Section 5.5. CTSs are subject to an increasing number of threats — we investigate security requirements and protection mechanisms in Section 5.6.

## 5.2 CONNECTED TRANSPORT SYSTEM APPLICATIONS AND WORKLOAD CHARACTERISTICS

The CVRIA reference architecture [1] identifies many connected vehicle applications in the categories: environmental, mobility, safety, and support services for Connected Vehicles. While the many use cases are constrained to traffic infrastructure, in-vehicle and vehicle-to-vehicle (V2V) infrastructure, an increasing number of use cases require backend capabilities. The aim of this section is to derive the requirements of CTS applications towards infrastructure.

In the following, we investigate application and workload characteristics of CTS data applications. Then, we use the identified characteristics to describe a subset of CVRIA applications.

- *Collection and Ingest*: A common challenge of CTS applications is the collection of data. Data needs to be collected from a diverse set of devices, it must be ingested into a Big data platform for refinement, processing, and analytics. To address security and privacy requirements data must be carefully treated, e.g., using data anonymization and masking techniques.
  - *Real time*: Data is loaded and analyzed as soon as it arrives. Often, this is done via a stream processing framework.
  - *Batch*: Incremental data loads in larger intervals (e.g., hourly, daily, or weekly).
- *Analytics*: Typical workloads include the parsing of large volumes of data into structured formats, such as columnar data format (Parquet, ORC), and aggregating these using abstractions, often as SQL and data frames. These applications may require multiple full passes of the data and joining/merging of multiple datasets. Location-based data requires capabilities for spatial analytics.
- *Machine Learning*: This involves the use of algorithms with the objectives of identifying patterns (unsupervised learning), classification and/or prediction (supervised learning). Challenges arise in particular when scaling machine learning to large volumes and high-dimensional datasets. More datasets related to connected transport are involving image and video data. In contrast to traditional analytics (based on SQL), machine learning often involves linear algebra operations on top of dense and even more challenging sparse arrays of features. Models can either be used for the purpose of data understanding or deployed in an online application.
- *Model Deployment*: Developed models are often deployed in an online system serving the user application. The coefficients for the scoring may be stored within a database system optimized for short-running and transactional queries. Systems that react to incoming data streams (e.g., event level data) require a streaming infrastructure for deployment.
  - *Short-Running Updates/Queries*: This mode is characterized by the usage of mainly short-running queries for lookups and updates on small volumes of data [3]. For this usage mode, the availability of mutable and random access storage is advantageous.
  - *Streaming*: Streaming applications process unbounded streams of incoming data and conduct incremental analytics on the data. An example application is traffic predictions. An increasing number of applications require both streaming and batch/interactive queries. The streaming phase is used for small-state updates and incremental analytics, while exploratory analytics, data modeling, and training are done in batch mode. Mutable storage simplifies the reconciliation of the batch and streaming layer.

From the use cases described in Chapter 1, the following critical infrastructure capabilities can be derived:

- Data is generated everywhere in CTSs—including the vehicle, the roadside infrastructure, and backend services. Depending on the available connectivity and bandwidth, an increasing amount of data can be centrally collected and made available for analysis and machine learning. In most cases, this data is ingested into the data lake. There it can serve for models supporting the actual use cases as well as secondary purposes, for example, predictive behavioral models for autonomous driving.
- Once loaded, the data is processed and refined using SQL and other tools. Analysis of geospatial data is an important capability for processing of location-based data. Machine learning models can have different complexities, from simple regression models based on a couple of hundred attributes to complex deep learning models.
- Model deployment: Current data and trained models are served to the front application using online systems optimized for short-running queries (e.g., HBase or a relational database). Models can be used for data understanding by deploying results in BI and Geo-Analysis tools or embedding them within an application. A challenge is to manage models deployed in the field and keep them up-to-date.
- Capabilities for real-time data processing that allow systems to learn from incoming data are increasingly important for a broad set of use cases in the domain of mobility. Low-latency use cases in the safety domain will increasingly rely on coordination capabilities in the backend requiring even lower latency than current state-of-the-art streaming infrastructures.

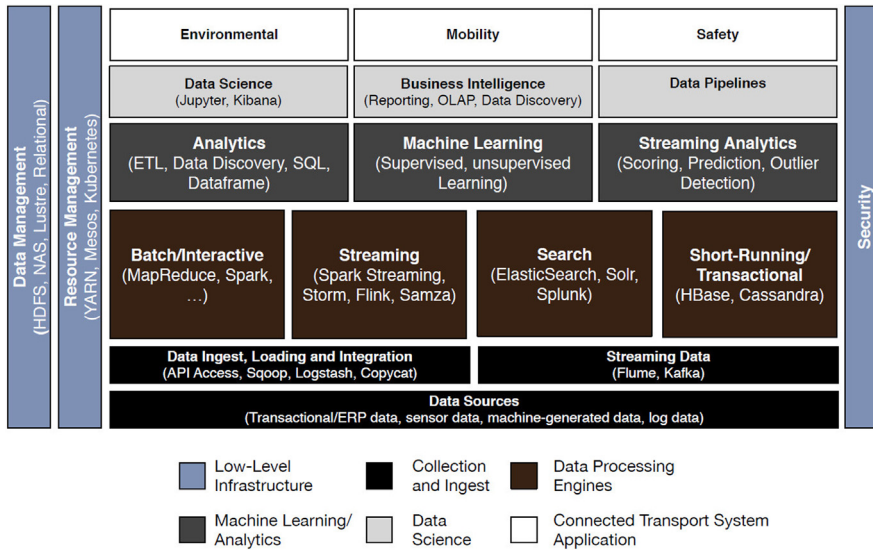
The complex requirements and processing patterns of CTS applications lead to complex infrastructure requirements. The landscape of data infrastructure is evolving rapidly which has resulted in many point solutions that often need to be connected. This typically leads to the necessity of moving data across diverse sets of storage backends, data formats, processing and serving frameworks that may result in complex data flows, the need for data integration/synchronization, and a high operational complexity. In the remainder of the chapter, we discuss the architecture of a Hadoop-based infrastructure to support these use cases and describe best practices for deploying this infrastructure and applications.

---

## 5.3 INFRASTRUCTURE OVERVIEW

To support the applications and the different stages of a data pipeline in a CTS, a data-centric infrastructure that enables data collection, storage, processing, and the ability to deploy models and serve results to data applications is required. We use the term *data lake* [4] to refer to such an infrastructure. A data lake is able to retain large volumes of data in its original, raw form to enable various kinds of analytics. The utilization of the data lake for advanced analytics and machine learning is referred to as *data science*.

Fig. 5.1 gives an overview of the infrastructure layers and components of a data lake for supporting connected transportation systems. Hadoop provides the core infrastructure for enabling this diverse set of applications. The following infrastructure layers can be identified.



**FIGURE 5.1** Infrastructure for connected transportation systems.

The architecture comprises six layers: In the application layer we have CTS applications in the three different domains that either access data analytics capabilities via data science or business intelligence (BI) environment or via embedded data pipelines. The machine learning layer provides high-level analytics capabilities for data modeling that are typically implemented using data processing engines (e.g., Spark for in-memory, iterative solvers). Data ingest is either done via batch or if real-time compute is required using streaming. Hadoop provides the data and compute capabilities required by all layers. Machine learning is described in detail in Chapter 12.

- *Low-Level Infrastructure:* The low-level infrastructure is responsible for managing compute and data resources. We are primarily concerned with Hadoop-based infrastructures, that is, HDFS for distributed storage and YARN for resource management. Frameworks on all levels interface with these base services when storing and processing data.
- *Data Collection and Ingest:* This layer incorporates technologies for connecting a wide variety of data feeds from CTS devices to the backend infrastructure. Often, this part of the infrastructure includes mechanisms for routing, message brokering, and data filtering/masking.
- *Data Processing Engines:* The data processing engine typically exposes a high-level abstraction, for example, MapReduce, and is responsible for mapping the application to low-level tasks (often using data parallelism), mapping and execution of these on a distributed infrastructure, and returning the results. Recent systems include higher-level abstractions, such as SQL, Search, and short-running requests.
- *Machine Learning and Analytics:* Libraries and frameworks that implement specific analytics functions, for example, a SQL execution engine or a specific machine learning algorithm, such as a logistic regression or a neural network.

- *Data Science*: Conduct iterative analysis on the data at hand, visualize data using business intelligence tools, and implement data pipelines to automate this process. Depending on the tool and platform this process is supported by visual interactions.
- *Connected Transport System Application*: Application classes as described by the CVRIA [1].

Each of the layers listed above is critical for a complete CTS. The bottom three layers provide the collection and processing of the streaming data that is ingested from connected vehicles. The top layers provide the analytics and machine learning needed for implementing use cases described in Table 5.1 such as Dynamic Eco Routing and Intelligent Traffic Signal Systems.

---

## 5.4 HIGHER-LEVEL INFRASTRUCTURE

This section focuses on the different programming systems, abstraction, and infrastructures required to support CTS applications.

### 5.4.1 MAPREDUCE AND BEYOND: SCALABLE DATA PROCESSING

The MapReduce [5] abstraction introduced by Google simplified the development of data-intensive applications. The abstraction consists of two main components: (i) a *map* function which processes a partition of the data and outputs a set of intermediate data in the form of a key–value pair; and (ii) the *reduce* function which performs an aggregate operation on the intermediate data. Between the *map* and the *reduce* functions, the framework sorts the data based on the key outputted in the map phase. While the abstraction is simplistic, it has proven suitable for many applications. In particular, the flexibility and simplicity (schema-on-read) of the MapReduce model contributed to its uptake. Further, the parallelization of MapReduce is well understood and can be automated at runtime without requiring the user to explicitly write a parallel code.

While various MapReduce implementations emerged, (e.g., Refs. [6,7]), Hadoop [2] is the most widely used open source application of the MapReduce abstraction. Hadoop has been used in different disciplines/domains for diverse data-intensive applications [8,9]. While the MapReduce abstraction was primarily designed to enable scalable data extractions, transformations and queries on large data volumes, it is increasingly used for implementing advanced analytics algorithms. A number of abstractions have been developed using MapReduce abstraction, mainly addressing the simplicity and ability to build more complex data flows. High-level languages (e.g., Pig [10], Cascading [11], Kite [12], and SpringXD [13]) provide a higher-level language and/or API that is then converted into MapReduce programs.

Hadoop MapReduce is based on a disk-oriented approach, that is after every MapReduce run data needs to be persisted in HDFS. This particularly results in slow access speeds for interactive or real-time analytics requiring queries and for iterative processing of machine learning. To address these issues, various processing and execution frameworks have emerged such as Spark [14], Flink [15] and Tez [16].

Spark has rapidly gained popularity. It utilizes in-memory computing, which makes it particularly suitable for iterative processing. The programming model is based on an abstraction referred

**Table 5.1 Selected Connected Transport System Use Cases [1] and Requirements: An Increasing Number of Use Cases Required a Sophisticated Backend and Data Processing Infrastructure**

Use Case	Collection and Ingest	Analytics and Machine Learning	Model Deployment
<i>Environmental:</i> Dynamic eco routing	Routing and fuel consumption data	Geospatial queries, exploratory data analysis, recommendation models	Serving latest predictions and recommendations for optimal route
<i>Environmental:</i> Road Weather information	Weather data (e.g., wiper speed and temperature)	Geospatial, exploratory, predictive	Serving latest weather forecast
<i>Mobility:</i> Electronic toll collection	Batch	Descriptive Analytics	Mainly transactional updates/queries
<i>Mobility:</i> Cooperative adaptive cruise control	Batch	Models for learning driving situations	Mainly V2V
<i>Mobility:</i> Vehicle data for traffic operations	Real time and batch	Descriptive and explorative. Models for predictions	Streaming models for predicting future infrastructure state
<i>Mobility:</i> Intelligent traffic signal system	Real time: Crowd-sources and infrastructure data	Geospatial, exploratory analytics/traffic light prediction models, deep learning for detecting traffic light states	Streaming for real-time model updates and calibration
<i>Mobility:</i> Dynamic ridesharing	Batch and real time	Descriptive analytics, match-making models, driver/rider scoring	Short-running queries for match-making. Streaming for update of models
<i>Mobility:</i> Smart park and ride system	Batch and real time	Geospatial and other analytics queries, prediction models	Streaming: Predict current parking situation in area
<i>Mobility:</i> Traveler information	Batch and real time: Park, travel information from various external systems and crowd-sourced	Geospatial and other analytics queries, prediction models	Streaming for model updates
<i>Safety:</i> Reduced speed Zone warning/lane closure/hazard warning	Real time	Geospatial and other analytics	Streaming and serving of warnings to vehicles in affected area
<i>Safety:</i> Blind spot/forward collision/lane change warning	Batch and real time	Driving behavior model	Mainly in-vehicle and V2V
<i>Safety:</i> Tailgating/Stationary/Slow Vehicle Advisory	Real time	Prediction models for impact prediction, driving behavior	In-vehicle and V2V for critical messages. Streaming to backend for coverage of larger areas

```

text_file = sc.textFile("hdfs://...")
counts = text_file.flatMap(lambda line: line.split(" ")) \
    .map(lambda word: (word, 1)) \
    .reduceByKey(lambda a, b: a + b)
counts.collect()

```

**LISTING 5.1**

Spark word count example.

to as *Resilient Distributed Datasets (RDD)*. RDDs are immutable, read-only data structures on which the application can apply transformations. The runtime handles the loading and distribution of the data into the cluster nodes' memory. The immutable nature of RDDs enables efficient recoveries after a failure by reapplying the transformations.

The Spark API was inspired by the MapReduce API; it is however much richer than the original MapReduce API. [Listing 5.1](#) shows a Spark word count example in Python. The example assumes an input file with text residing in the Hadoop filesystem. The Spark API defines transformations and actions. Transformations can be chained to each other. They are not evaluated until an action is called (in the word count example `collect()`). The call to `collect` executed the chained transformation: `flatMap` splits the line into words, `map` emits every word and the number “1” (`(word, 1)`), and `reduceByKey` aggregates the “1”s for every word.

A constraint of Spark is that its in-memory capabilities are limited to single application jobs. There are multiple ongoing efforts to extract in-memory capabilities into a separate runtime layer that is usable by multiple frameworks and not restricted to caching data within a single framework and/or job. Tachyon [\[17\]](#) is an example of such a distributed in-memory filesystem, which is based on HDFS as the underlying filesystem.

Another important requirement for CTS applications is having the ability to process geospatial data. Spatial extensions for relational databases, such as Oracle and Postgres, exist, but do not provide the scale necessary for large-volume use cases. Magellan [\[18\]](#) is a spatial library that simplifies the processing of spatial data using Spark.

## 5.4.2 DATA INGEST AND STREAM PROCESSING

Traditional approaches for datasets focus on the collection, storage, and analysis of complete, bounded datasets. In the CTS context, it is often critical to analyze data as it arrives potentially at the edge of the network or on the device itself. The landscape of tools and frameworks for supporting this kind of stream processing is manifold (see Ref. [\[19\]](#) for survey). The majority of these tools are open source and emerged in the Hadoop ecosystem. In the following text, we briefly highlight the main components for stream processing: the message brokering system, the storage, and the actual stream processing engine (see Ref. [\[20\]](#)).

*Message Broker:* The message broker decouples data production (transportation system sensors, connected vehicles, etc.) and consumptions; Typically, message brokers are implemented as Publish/Subscribe systems that enable data producer to publish data and consumer to process this data asynchronously and at its own pace while ensuring consistency, durability, and fault tolerance. An example of a message broker is Kafka [\[21\]](#). Kafka is a distributed system comprising of multiple message brokers coordinated via a Zookeeper cluster designed to support large volumes of data



(in particular log files and telematics data). Similar capabilities are provided by RabbitMQ which provides more message routing options and delivery guarantees, but is less scalable than Kafka. Increasingly, message broker capabilities are also offered as Platform-as-a-Service cloud offering, e.g., Amazon Kinesis and Google Cloud Pub-Sub.

*Processing Engines for Streaming:* Various stream processing engines merged Flink [15], Spark Streaming [22], Samza [23] and Storm [24], and its successor Heron [25]. There are two types of streaming engines: native stream engines, e.g., Flink, continuously process incoming data, while micro-batch engines, e.g., Spark Streaming, accumulate incoming data into small batches. Apache Beam [29] and its cloud implementation: Google's Dataflow [28] are further examples for native stream processing engine. The Beam API provides a rich abstraction for implementing stream and batch data flows. The core of the API is a well-defined semantic for specifying processing windows.

### 5.4.3 SQL AND DATAFRAMES

While Hadoop MapReduce and Spark provide a set of well-defined, scalable abstractions for efficient data processing, higher-level abstractions to structured data, such as SQL, enable higher productivity. SQL has proven to be a resilient method of querying data. The advantages of SQL are that it is widely known, and its query language provides a robust way by which to wrangle data. Many use cases rely on SQL as a common grammar for data extraction. It is particularly useful for querying columnar data that has been derived from less structured data sources. In general, two architectures have emerged: (i) the integration of Hadoop with existing relational datawarehouse systems and (ii) the implementation of SQL engines on top of core Hadoop services, that is, HDFS and YARN. Architecture (i) is often used to integrate data from Hadoop into an existing datawarehouse. The scalability of this architecture is limited, as queried data always needs to be processed in the database (which is typically a magnitude smaller than the Hadoop cluster). In the following text, we focus on Hadoop SQL engines.

Inspired by Google's Dremel [28], various SQL query engines running over Hadoop have emerged: Hive [29], HAWQ [32], Impala [33], and Spark SQL [34]. One of the first and still very widely used SQL engines is Hive. Early versions of Hive compiled SQL into MapReduce jobs that often yielded nonoptimal performance. Thus, Hive was extended to multiple runtimes, for example, Tez [16] and Spark. Tez generalizes the MapReduce model to a generic dataflow-oriented processing model while providing an improved runtime system that supports in-memory caching and container reuse.

While SQL performs well for data queries and transformations, for machine learning, more flexible abstractions are desirable. The *dataframe* abstractions originally introduced in R expose data in a tabulated format to the user and support the efficient expression of data transformations and analytics [33]. A dataframe typically stores a matrix of different types of data (e.g., numeric, text data and categorical data). The dataframe abstraction supports various functions for manipulating data stored inside the data structures, for example, to subset, merge, filter and aggregate data. Similar abstractions emerged for other languages, for example, Pandas [34], Scikit-Learn [35] and SFrame [36] for Python.

The most-well known Hadoop-based dataframe implementation is Spark dataframes. In contrast to R or Pandas, Spark Dataframes are stored in distributed memory across multiple nodes in the



clusters. Dataframes are based and tightly integrated with Spark SQL and enable users to combine different programming models for data extraction and feature engineering. Recently, extensions to the dataframe abstractions have been proposed; GraphFrames [37] is a higher-level abstraction based on Spark Dataframes for representing graphs and expressing queries on graphs.

Dataframes are a powerful abstraction for data manipulation, analysis, and modeling. To move ad-hoc data analytics to production, the different analytic steps need to be connected in an end-to-end application. Frameworks that provide a pipeline API are Scikit-Learn's, Spark MLlib [38], and KeystoneML [39]. The spark.ml API of MLLib provides two fundamental abstractions for expressing data manipulations: *transformers* for augmenting datasets and *estimators* for learning a model. These abstractions are typically combined in a Pipeline.

#### 5.4.4 SHORT-RUNNING AND RANDOM ACCESS DATA MANAGEMENT

Most Hadoop tools rely on fast sequential reads that enable scalable analytics applications. Other data access modes, for example, for short-running and random access queries, have only been a second-order concern in contrast to traditional relational database systems. For example, HBase allows mutable and random access datasets. HBase [40] is a Hadoop-based column-oriented datastore based on the HDFS filesystem. Other Hadoop-based analytics frameworks (e.g., Hive and Spark) can directly access it without the need to move the data.

#### 5.4.5 SEARCH-BASED ANALYTICS

Gartner defines search-based data discovery tools as tools that allow end/business users to create views and analyses of structured and unstructured data applying search terms [41]. Several tools for aiding search-based data discovery emerged, for example, Elasticsearch, Solr, and Splunk. The ELK stack refers to the usage of three complementary open source tools: Elasticsearch [42], Logstash [43] and Kibana [44]. Elasticsearch supports search-based data analytics based on the abstraction of an index, Logstash is a data ingestion and enrichment tool primarily designed for log files, and Kibana is a visualization tool.

#### 5.4.6 BUSINESS INTELLIGENCE AND DATA SCIENCE

Visualization is a key part of the data analysis process and is critical for providing insight into the analytics. Two sets of tools emerged for supporting data analysis: BI tools typically focused on the ability to create dashboards on top of well-known and structured data sources. Data science tools support deeper kinds of data processing and complex data pipelines for cleaning, preparing, and analyzing data. For this purpose, access of a wide range of data sources from Excel files to Hadoop clusters and to relational databases is necessary. Both tool categories are converging as BI tools add capabilities for accessing Hadoop clusters and execute advanced analytics (e.g., by integrating R). At the same time new visual tools for data exploration and discovery emerged, for example, Trifacta.

Examples of BI tools are Tableau [45] and Qlik [46]. Further, cloud-based BI tools, such as Microsoft's Power BI [47] and Amazon's QuickSight [48], are becoming increasingly important in particular as more data is generated, collected, and stored in public cloud environments. Data science tools are oriented around the concept of a notebook, a page that combines interactive code execution,

visualizations, and documentation. Jupyter/iPython [49] is one of the used notebooks for data analysis supporting code written in Python, Julia, Lua, R, and many other languages. Jupyter notebooks can seamlessly integrate with visualizations done using Matplotlib [50], Seaborn [51], Bokeh [52], and ggplot2 [53]. Further examples of notebook environments are Apache Zeppelin [54]. Also, there are emerging cloud-based notebooks, for example, the Databricks cloud [55] that is based on Spark.

### 5.4.7 MACHINE LEARNING

Most data science involves the usage of hundred mostly hand-curated features and simple well-understood algorithms, such as linear and logistic regressions, support vector machines, random forest, etc. Both R and Python provide a rich set of libraries for machine learning. The Python data ecosystem comprises powerful scientific and analytics libraries, such as NumPy [56], Pandas [34], and Scikit-Learn [57]. However, these are typically not parallelized and are thus constrained in their scalability. Mahout [58], MLlib [38], Dato [59], and H2O [60] are some examples that provide high-level machine learning capabilities on top of Hadoop.

The majority of data generated in CTS environments is unstructured, for example, video, text, sensor, and image data. The usage of unstructured, high-dimensional data (e.g., images and text) requires more complex modeling approaches, such as topic modeling and deep learning, which have more complex infrastructure requirements. The term *deep learning* is used for a large class of neural network-based machine learning models [61]. Neural networks are modeled after the human brain and use multiple layers of neurons—each taking multiple inputs and generating an output—to fit the input to the output.

Neural networks are available in many machine learning libraries for different languages, for example, Pylearn2, Theano [62], Java/DL4J [63], R/neuralnet [64], Caffe [65], Tensorflow [66], Microsoft CNTK [67], and Lua/Torch [68]. The ability to customize these networks, for example, by adding and customizing layers, differs. While some libraries, such as Pylearn, focus on a high-level, easy-to-use abstractions for deep learning frameworks (e.g., Tensorflow, Theano and Torch) are highly customizable and optimized for development of custom networks.

Neural networks—in particular deep networks with many hidden layers—are challenging to scale. Also, the application/scoring against the model is more compute intensive than for other models. GPUs have been proven to scale neural networks particularly well but have their limitations for larger image sizes. Several libraries rely on GPUs for optimizing the training of neural networks [69], for example, NVIDIA's cuDNN [70], Theano [71] (used by Pylearn), and Torch. Currently, only a few distributed/parallel implementations of neural networks exist—for example, Google's DistBelief [72] (not publicly available) and H2O [60]. Hybrid architecture using a cluster of GPU-enabled nodes is under development, for example, by Baidu [73]. More discussion on Machine Learning is in Chapter 12.

---

## 5.5 LOW-LEVEL INFRASTRUCTURE

This section focuses on the storage and compute management of Hadoop and running Hadoop in the cloud—necessary for a scalable infrastructure that can handle CTS data volume and velocity.

### 5.5.1 HADOOP: STORAGE AND COMPUTE MANAGEMENT

The Hadoop core comprises two components: the Hadoop Distributed File System (HDFS) [74] and Yet Another Resource Negotiator (YARN) [75]. HDFS provides a distributed filesystem that is capable of scaling out as the size of the data increases while providing redundancy and integrity. As the amount of automotive data is expected to increase to 11.1 PB per year by 2020, this flexibility in scale becomes critical. YARN provides resource management for the cluster. A typical CTS cluster will have various services running simultaneously. These may include Sqoop jobs ingesting relational data, Kafka and/or Spark streaming jobs collecting real-time traffic and connected vehicle data, SQL jobs performing analytics on the traffic data (Spark SQL, Hive, Impala), and machine learning algorithms (Spark MLlib, Mahout) running across collections of CTS datasets. The memory and compute resources for each node are handled by YARN in order to determine how to best allocate these resources to the different tasks.

Traditionally, Hadoop has been deployed on commodity hardware enabling highly cost-efficient environments. However, there are several challenges associated with this approach namely the high management complexity of such environments that so far could not be entirely addressed by enterprise add-ons, such as Cloudera Manager. Thus, there is an increasing interest in alternate approaches, for example, running Hadoop on top of other parallel filesystems, such as Lustre, or storage appliances like EMC Isilon, the usage of Hadoop appliances and clouds.


### 5.5.2 HADOOP IN THE CLOUD

While the majority of data infrastructures are deployed on-premise in enterprise data centers, increasingly novel infrastructure delivery methods are being used. There are three models that may be used with cloud computing: public, private, and hybrid. Each of these models has both advantages and disadvantages. The best model will depend upon the need to scale workloads, performance, security, and cost. Table 5.2 provides details about the different deployment considerations for Hadoop-based infrastructures.

Public clouds provide the most flexibility in scale. This is particularly important with CTSs due to the nature of traffic data. Transportation data has peak times of the data, and certain days such as holidays, may result in larger than normal data requests. A public cloud is able to scale to meet this demand. The most common method of deploying Hadoop in the cloud are Infrastructure as a Service setups, which is supported by various tools, for example, Apache Whirr, Cloudera Directory, or Hortonworks Cloudbreak. Further, there are several higher-level services for Hadoop and Spark available—Elastic MapReduce and Microsoft’s HDInsight. Fig. 5.2 illustrates the usage of the AWS Elastic MapReduce portal for setting up a ready to use Hadoop cluster. The user can select from various configurations and define the size of the cluster meeting the dataset’s processing requirements. In a typical cloud deployment data is retained in block storage services (e.g., Google Storage, Azure Blob Storage, and Amazon’s S3) and moved to the Hadoop cluster for processing.

However, long-running Hadoop clusters in the cloud are often not cost efficient. Thus, data often needs to be pushed to cheaper object stores like Amazon S3 [76], Google Cloud Storage [77], or Azure Blob Storage [78]. While an increasing number of frameworks such as Spark and Impala optimize for object storage backends, there is a performance trade-off associated with these architectures that will have an effect on the analytics and potentially the machine learning workloads.

	On-Premise	HPC	Private	Public
<i>Architecture</i>	Dedicated Hard-ware	Dedicated hardware managed by HPC scheduler (SLURM, PBS)	Manually setup VM clusters	Fully-managed VM cluster (manual setup possible)
<i>Maturity</i>	High	Medium	Medium	Medium
<i>Flexibility/ extensibility/ elasticity</i>	Low	Low	Low	High (cluster can be dynamically expanded/ shrunk)
<i>Performance</i>	Optimal	Optimal	Good	Good
<i>Security</i>	High	High	High	Medium
<i>Management tools</i>	Ambari, Cloudera Manager	saga-Hadoop, jump	Apache Whirr, Cloudbreak	AWS Elastic MapReduce, Azure HDInsights

 **AWS** ▼ **Services** ▼ **Edit** ▼

Create Cluster - Quick Options [Go to advanced options](#)

General Configuration

Cluster name

CTS Cluster

☒ Logging ⓘ

S3 folder

Launch mode

☒ Cluster ⓘ ☐ Step execution ⓘ

Software configuration

Vendor

☒ Amazon ☐ MapR

Release

emr-5.0.0 ⓘ

Applications

☒ Core Hadoop: Hadoop 2.7.2 with Ganglia 3.7.2, Hive 2.1.0, Hue 3.10.0, Mahout 0.12.2, Pig 0.16.0, and Tez 0.8.4

☐ HBase: HBase 1.2.2 with Ganglia 3.7.2, Hadoop 2.7.2, Hive 2.1.0, Hue 3.10.0, Phoenix 4.7.0, and ZooKeeper 3.4.8

☐ Presto: Presto 0.150 with Hadoop 2.7.2 HDFS and Hive 2.1.0 Metastore

☐ Spark: Spark 2.0.0 on Hadoop 2.7.2 YARN with Ganglia 3.7.2 and Zeppelin 0.6.1

Hardware configuration

Instance type

m3.xlarge

Number of instances

3 (1 master and 2 core nodes)

**FIGURE 5.2**  
AWS Elastic MapReduce.

As mentioned, SQL has proven to be resilient and is frequently used in data analytics. Hadoop-based SQL engines such as Hive, Impala, and Spark SQL are capable of handling terabytes of data. However, as the CTS data size increases to petabytes of information, the ability of these systems to analyze the data in a reasonable amount of time degrades — even with scaling out a cluster. Instead, MPP (Massively Parallel Processing) databases must be used to handle such datasets without sacrificing performance. Two such databases are Amazon’s Redshift [79] and Google’s BigQuery [80]. These provide a completely managed and elastic query engine. MPP databases running on public, private, and/or hybrid clouds will be necessary for the projected growth in connected transportation data.

---

## 5.6 CHAPTER SUMMARY AND CONCLUSIONS

Infrastructures for CTSs need to be designed to meet a complex set of requirements from a diverse set of applications. An increasing number of these applications is data- and backend-driven, for example, the majority of use cases in the CVRIA mobility domain: intelligent traffic signal systems, smart parking, and travel information systems. The use cases require a secure and scalable infrastructure capable of dealing with real-time data streams while supporting complex machine learning models on large volume historical datasets. A Hadoop-based infrastructure—a data lake—can address many of these requirements. The Hadoop ecosystem and open source community provides a manifold set of tools for data wrangling, SQL, machine learning, and data streaming. Streaming capabilities are particularly important for CTS as the ability to react to incoming data in near real time is critical.

This chapter discussed data infrastructure to support Connected Transportation System applications. It provided an overview of infrastructures to support the requirements of data infrastructure capable of storing, processing, and distributing large volumes of data using different abstractions and runtime systems. Hadoop, a scalable platform for compute and storage, has been extensively used for Big Data processing at Internet companies and in the scientific community. A vibrant ecosystem of tools for data processing, advanced analytics, and machine learning exists. In this ecosystem, Spark and Hadoop provide the core infrastructure for storage, batch, and stream processing. In the future, we expect that the majority of data processing will happen in the cloud. The three major cloud platforms: (i) Amazon Web Services, (ii) Google, and (iii) Microsoft Azure, provide various options for addressing the need for connected vehicle data processing.

In the future, as CTSs become more data-driven, the needs for stream processing will increase. Further, the complexity of data will increase, for example, due to the increasing deployment of camera-based sensors. Thus, emerging deep learning techniques, like convolutional neural networks, will become essential. Deep learning demands new infrastructure components, for example, accelerators, GPUs etc.

---

## EXERCISE PROBLEMS AND QUESTIONS

1. Explain the MapReduce abstraction! What are the benefits of using MapReduce? What is the advantage of Spark over the traditional Hadoop MapReduce? Illustrate the trade-offs between Spark RDD and higher-level abstractions like SQL and Dataframes!

2. What are the trade-offs between the different deployment options of CTS Infrastructures (Cloud vs. on-premise)?
3. Setup a small Hadoop instance using either AWS or a local Hortonworks/Cloudera Virtual Machine. The instance should have Hive and Spark for data processing and machine learning.
4. A central component of a CTS is Vehicle-to-Vehicle (V2V) communication. As of this writing, no mass produced car currently supports V2V; however, support is planned for the 2017 Cadillac CTS and Mercedes-Benz E-Class.
5. If you were to buy one of these models, with what frequency would your vehicle be able to use V2V communication each day (this would be dependent upon the number of vehicles you encounter on your drive and the sales numbers for these models in your region).
6. A certain percentage of cars on the road must have V2V communication in order to make the technology effective. If we have a use case that requires 15% of the cars in our region to have V2V, and we would like to be able to perform this use case by 2022, what percentage of new cars each year will need to have V2V? This will need to take into account new vehicle sales ( $\sim 7.5$  million per year), and the average age of the fleet ( $\sim 11.4$  years in the United States). The output can be a graph showing the growth in V2V cars.
7. The Road Weather Information use case can utilize national/regional datasets as well as data from nearby vehicles. Utilization of the former is already done in a significant number of vehicles in which Connected car systems report on the weather based upon GPS coordinates. Download an hourly weather dataset into Hadoop as well as vehicle location data using GPS coordinates and time (can be a synthetic dataset). Create a Hive or Spark SQL query to return the correct hourly weather forecast for each requesting vehicle.
8. The Traveler Information use case utilizes external systems as well as crowd-sourcing to provide travel information. Download an hourly traffic dataset that can be combined with the weather dataset from Exercise 3. Make a prediction utilizing R, Spark MLlib, or Scikit-Learn about what the traffic is likely to be for the next 30 days given past traffic information and weather patterns.

---

## REFERENCES

- [1] US Department of Transportation, Connected Vehicle Reference Implementation Architecture. <<http://www.iteris.com/cvria/>>, 2014.
- [2] Hadoop: Open Source Implementation of MapReduce. <<http://hadoop.apache.org/>>.
- [3] P. Bailis, J. M. Hellerstein, M. Stonebraker, (Eds), Readings in Database Systems, fifth ed., 2015.
- [4] N. Heudecker, M.A. Beyer, L. Randall, Defining the Data Lake., Gartner Research, 2015. <<https://www.gartner.com/document/3053217>>, Gartner, Inc., Stamford, CT.
- [5] J. Dean, S. Ghemawat, MapReduce: Simplified data processing on large clusters, OSDI'04: Proceedings of the 6th conference on Symposium on Operating Systems Design & Implementation, USENIX Association, Berkeley, CA, USA, 2004, pp. 137–150.

- [6] P.K. Mantha, A. Luckow, S. Jha, Pilot-MapReduce: An extensible and flexible MapReduce Implementation for distributed data, *Proceedings of third international work-shop on MapReduce and its Applications, MapReduce '12*, ACM, New York, NY, USA, 2012, pp. 17–24.
- [7] M. Isard, M. Budiu, Y. Yu, A. Birrell, D. Fetterly, Dryad: Distributed data—parallel programs from sequential building blocks, *SIGOPS Oper. Syst. Rev.* 41 (3) (March 2007) 59–72.
- [8] A. Luckow, K. Kennedy, F. Manhardt, E. Djerekarov, B. Vorster, A. Apon, Automotive big data: Applications, workloads and infrastructures, *Proceedings of IEEE Conference on Big Data*, IEEE, Santa Clara, CA, USA, 2015.
- [9] J.L. Hellerstein, K. Kohlhoff, D.E. Konering, Science in the cloud, *IEEE Internet Computing* 16 (4) (2012) 64–68.
- [10] C. Olston, B. Reed, U. Srivastava, R. Kumar, A. Tomkins, Pig latin: A not-so-foreign language for data processing, *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, ACM, New York, NY, USA, 2008, pp. 1099–1110.
- [11] Cascading. <<http://www.cascading.org/>>, 2016.
- [12] Kite: A Data API for Hadoop. <<http://kitesdk.org/>>, 2016.
- [13] Spring X.D. <<http://projects.spring.io/spring-xd/>>, 2016.
- [14] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, et al., Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing, *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, USENIX Association, Berkeley, CA, USA, 2012 pp 2–2.
- [15] A. Alexandrov, R. Bergmann, S. Ewen, J.-C. Freytag, F. Hueske, A. Heise, et al., The stratosphere platform for big data analytics, *VLDB J.* 23 (6) (December 2014) 939–964.
- [16] Apache tez. <<http://tez.apache.org/>>, 2016.
- [17] H. Li, A. Ghodsi, M. Zaharia, E. Baldeschwieler, S. Shenker, and I. Stoica. Tachyon: Memory throughput i/o for cluster computing frameworks. <[https://amplab.cs.berkeley.edu/wp-content/uploads/2014/03/2013\\_ladis\\_tachyon1.pdf](https://amplab.cs.berkeley.edu/wp-content/uploads/2014/03/2013_ladis_tachyon1.pdf)>, 2013, *Proceedings of LADIS 2013*.
- [18] R. Sriharsha. Magellan: Geo Spatial Data Analytics on Spark. <<https://github.com/harsha2010/magellan>>, 2015.
- [19] S. Kamburugamuve, G. Fox, Survey of distributed stream processing, Technical report, Indiana University, Bloomington, IN, USA, 2016.
- [20] A. Luckow, P.M. Kasson, and S. Jha, Pilot-streaming: Design considerations for a stream processing framework for high-performance computing. White Paper submitted to STREAM16, 2016.
- [21] G. Wang, J. Koshy, S. Subramanian, K. Paramasivam, M. Zadeh, N. Narkhede, et al., Building a replicated logging system with apache kafka, *PVLDB* 8 (12) (2015) 1654–1665.
- [22] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, I. Stoica, Discretized streams: Fault-tolerant streaming computation at scale, *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, SOSP '13, ACM, New York, NY, USA, 2013, pp. 423–438.
- [23] M. Kleppmann, J. Kreps, Kafka, Samza and the Unix philosophy of distributed data, *IEEE Data Engineering Bulletin* (December 2015) Journal Article.
- [24] Twitter. Storm: Distributed and fault-tolerant realtime computation. <<http://storm-project.net/>>.
- [25] S. Kulkarni, N. Bhagat, M. Fu, V. Kedigehalli, C. Kellogg, S. Mittal, et al., Twitter heron: Stream processing at scale, *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, SIGMOD '15, ACM, New York, NY, USA, 2015.
- [26] Apache beam proposal. <<https://wiki.apache.org/incubator/BeamProposal>>, 2016.
- [27] T. Akidau, R. Bradshaw, C. Chambers, S. Chernyak, R.J. Fernandez-Moctezuma, R. Lax, et al., The dataflow model: A practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing, *Proc. VLDB Endow.* 8 (2015) 1792–1803.



- [28] S. Melnik, A. Gubarev, J.J. Long, G. Romer, S. Shivakumar, M. Tolton et al., Dremel: Interactive analysis of web-scale datasets. In Proc. of the 36th Int'l Conf on Very Large Data Bases, 2010, pp. 330–339.
- [29] A. Thusoo, J.S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, et al., Hive: A warehousing solution over a map-reduce framework, Proc. VLDB Endow. 2 (2) (August 2009) 1626–1629.
- [30] M.A. Soliman, L. Antova, V. Raghavan, A. El-Helw, Z. Gu, E. Shen, et al., Orca: A modular query optimizer architecture for big data, *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, ACM, New York, NY, USA, 2014, pp. 337–348.
- [31] M. Kornacker, A. Behm, V. Bittorf, T. Bobrovitsky, C. Ching, A. Choi, et al., Impala: A modern, open-source sql engine for hadoop. In CIDR. <[www.cidrdb.org](http://www.cidrdb.org)>, 2015.
- [32] M. Armbrust, R.S. Xin, C. Lian, Y. Huai, D. Liu, J.K. Bradley, et al., Spark SQL: relational data processing in spark, in: T. Sellis, S.B. Davidson, Z.G. Ives (Eds.), *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, Melbourne, Victoria, Australia, May 31- June 4, 2015, ACM, 2015, pp. 1383–1394.
- [33] R Core Team, R: A Language and Environment for Statistical Computing, R Foundation for Statistical Computing, Vienna, Austria, 2013 ISBN 3-900051-07-0.
- [34] W. McKinney. Data structures for statistical computing in python, In: S. van der Walt and J. Millman (Eds.), *Proceedings of the 9th Python in Science Conference*, 2010, pp. 51–56.
- [35] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, et al., API design for machine learning software: Experiences from the scikit-learn project. CoRR, abs/1309.0238, 2013.
- [36] SFrame: Scalable tabular and graph data structures, 2016.
- [37] GraphFrames Package for Apache Spark. <<http://graphframes.github.io/>>, 2016.
- [38] X. Meng, J.K. Bradley, B. Yavuz, E.R. Sparks, S. Venkataraman, D. Liu, et al., Mllib: Machine learning in apache spark, CoRR (2015) abs/1505.06807.
- [39] E. Sparks and S. Venkataraman. KeystoneML. <<http://keystone-ml.org/>>, 2016.
- [40] D. Borthakur, et al., Apache Hadoop goes realtime at facebook, *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, ACM, New York, NY, USA, 2011, pp. 1071–1080.
- [41] Search-Based Data Discovery Tools. <<http://www.gartner.com/it-glossary/search-based-data-discovery-tools>>, 2016.
- [42] Elastic Search. <<https://github.com/elastic/elasticsearch>>, 2016.
- [43] Logstash. <<https://github.com/elastic/logstash>>, 2016.
- [44] Kibana. <<https://github.com/elastic/kibana>>, 2016.
- [45] Tableau. <<http://www.tableau.com>>, 2016.
- [46] Qlikview and qlik sense. <<http://www.qlik.com/>>, 2016.
- [47] Microsoft. Power bi. <<https://powerbi.microsoft.com/>>, 2016.
- [48] Amazon. Quicksight. <<https://aws.amazon.com/quicksight/>>, 2016.
- [49] F. Perez, B.E. Granger, Ipython: A system for interactive scientific computing, *Comput. Sci. Eng.* 9 (3) (2007) 21–29.
- [50] J. D. Hunter, Matplotlib: A 2d graphics environment. *Comput. Sci. Eng.* 9(3) (2007) 90–95.
- [51] M. Waskom. Seaborn: statistical data visualization. <<https://stanford.edu/~mwaskom/software/seaborn/>>, 2015.
- [52] Bokeh Development Team. Bokeh: Python library for interactive visualization. <<http://www.bokeh.pydata.org>>, 2014.
- [53] H. Wickham, ggplot2: Elegant Graphics for Data Analysis, Springer-Verlag, New York, 2009.
- [54] Apache zeppelin (incubating). <<https://zeppelin.incubator.apache.org/>>, 2015.
- [55] Databricks. <<https://databricks.com/>>, 2016.

- [56] S. van der Walt, S.C. Colbert, G. Varoquaux, The numpy array: A structure for efficient numerical computation, *Comput. Sci. Eng.* 13 (2) (2011) 22–30.
- [57] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, et al., Scikit-learn: Machine learning in Python, *J. Machine Learning Res.* 12 (2011) 2825–2830.
- [58] Apache Mahout. <<http://mahout.apache.org/>>, 2014.
- [59] D. Bickson. Dato’s Deep Learning Toolkit. <<http://blog.dato.com/deep-learning-blog-post>>, 2015.
- [60] H2O – Scalable Machine Learning. <<http://h2o.ai/>>, 2015.
- [61] T.J. Hastie, R.J. Tibshirani, J.H. Friedman, The elements of statistical learning: data mining, inference, and prediction, Springer series in statistics, Springer, New York, 2009.
- [62] I.J. Goodfellow, D. Warde-Farley, P. Lamblin, V. Dumoulin, M. Mirza, R. Pascanu, et al., Pylearn2: A machine learning research library. *arXiv preprint arXiv:1308.4214*, 2013.
- [63] Deep Learning for Java. <<http://deeplearning4j.org/>>, 2015.
- [64] F. Günther and S. Fritsch. Neuralnet: Training of neural networks. *R J.* 2(1) (jun 2010) 30–38.
- [65] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R.B. Girshick, et al., Caffe: Convolutional architecture for fast feature embedding, *CoRR* (2014) abs/1408.5093.
- [66] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.
- [67] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *ArXiv e-prints*, December 2015.
- [68] R. Collobert, K. Kavukcuoglu, and C. Farabet, Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, number EPFL-CONF-192376, 2011.
- [69] N. Vasilache, J. Johnson, M. Mathieu, S. Chintala, S. Piantino, Y. LeCun, Fast convolutional nets with fbfft: A GPU performance evaluation, *CoRR* (2014) abs/1412.7580.
- [70] NVIDIA cuDNN. <<https://developer.nvidia.com/cuDNN>>, 2015.
- [71] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio. Theano: A cpu and gpu math expression compiler. In *Proceedings of the Python for scientific computing conference (SciPy)*, volume 4, page 3. Austin, TX, 2010.
- [72] J. Dean, G.S. Corrado, R. Monga, K. Chen, M. Devin, Q.V. Le, et al., Large scale distributed deep networks. In *NIPS*, 2012.
- [73] R. Wu, S. Yan, Y. Shan, Q. Dang, G. Sun, Deep image: Scaling up image recognition, *CoRR* (2015) abs/1501.02876.
- [74] K. Shvachko, H. Kuang, S. Radia, R. Chansler, The Hadoop distributed file system, *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, MSST ’10, IEEE Computer Society, Washington, DC, USA, 2010, pp. 1–10.
- [75] V.K. Vavilapalli. Apache Hadoop YARN: Yet Another Resource Negotiator. In *Proc. SOCC*, 2013.
- [76] Amazon S3 Web Service. <<http://s3.amazonaws.com>>.
- [77] Google Cloud Storage. <<https://developers.google.com/storage/>>.
- [78] Windows Azure Blob Storage. <<https://azure.microsoft.com/en-us/documentation/services/storage/>>.
- [79] Amazon redshift. <<https://aws.amazon.com/redshift/>>, 2016.
- [80] Google Big Query. <<https://developers.google.com/bigquery/docs/overview>>.