ETL Data Pipelines Configurations in Spark

BACHELOR THESIS

Gizem Batmaci

Submitted on 20 September 2022

Friedrich-Alexander-Universität Erlangen-Nürnberg
Faculty of Engineering, Department Computer Science
Professorship for Open Source Software

Supervisor:
Prof. Dr. Dirk Riehle, M.B.A.
Reza Ghanbari, M.Sc.

Friedrich-Alexander-Universität
Faculty of Engineering

Declaration of Originality

I confirm that the submitted thesis is original work and was written by me without further assistance. Appropriate credit has been given where reference has been made to the work of others. The thesis was not examined before, nor has it been published. The submitted electronic version of the thesis matches the printed version.

Erlangen, 20 September 2022

License

Erlangen, 20 September 2022

Abstract

*T*he JValue Open Data Service (ODS) is an ETL data pipeline that provides data extraction from different source systems (Extract), performs transformations on the extracted data (Transform), and loads the data to a target database (Load). There are different kinds of stream processing engines that cope with data that have high volume, variety, and velocity. Existed ETLs cannot be applied to different streaming services, and the use of various frameworks and programming languages brings complexity along. Among different streaming services, Apache Spark offers accelerated, reusable, and scalable ETLs. This thesis aims to suggest an approach to compile and configure a data pipeline and have it runnable on Apache Spark.

Contents

List of Figures

List of Tables

Acronyms

API Application Programming Interface

CSV Comma Separated Values

DSL Domain-Specific Language

IDE Integrated Development Environment

ETL Extract, Transform, and Load

GUI Graphical User Interface

SDK Software Development Kit

SQL Structured Query Language

JSON JavaScript Object Notation

1 Introduction

The term ETL stands for Extract, Transform and Load and it integrates different but interrelated steps to process data. An ETL data pipeline extracts data from heterogeneous sources such as SQL Server, MySQL, Oracle, SQL Server, DB2, Hana databases. After extraction stage, ETL performs transformations to convert data into the necessary and desired format by cleaning, deriving new calculations, applying a set of validations, splitting, merging, aggregating, and joining. In the load stage, transformed data is loaded to a target system, generally a data warehouse or a database (Raghuraman & College, 2021). Referential integrity maintenance and efficient resource use are crucial for data loading stage.

There are different popular commercial, open source, cloud-based, and custom ETL tools: Informatica, Talend, Google Cloud Dataflow, IBM, AWS Glue, Azure Data Factory, Oracle Data Integrator, SAS Data Management,Talend Open Studio, and Hadoop. Due to the high complexity and time-consuming applications of ETL, 80% of the time is estimated to be devoted to a data warehouse project (Kuchen et al., 2011). There are also different kinds of code-based and GUI-based ETL tools available. Code-based ETL solutions are generally faster, cheaper and easier-to-maintain than GUI-based ETL pipelines. GUI-based ETL pipelines are easy to use with visual approaches for people who does not have programming knowledge. Yet, the capabilities of GUI-based ETL tools mostly fail to satisfy specific ETL scenarios. Therefore, code-based ETL tools are more productive and flexible to implement complex flows rather than GUI-based ETL tools designed with drag and drop function (Pall & Khaira, 2018).

Traditional ETL tools are mainly used for data migrating from various sources to one target system. They target collecting all data in one system in order to perform deeper and comprehensive data analysis, draw insights from data, and make estimations based on high volume of data. Yet, they bring along some problems: They are not able to process data that have high volume, velocity, and variety. Diversity in data requires changes on ETLs on a regular basis but on the other hand ETLs are generally not reusable (Mukherjee & Kar, 2017).

Various big data frameworks and streaming engines such as Apache Spark, Apache Storm, Apache Kafka, Apache Pulsar, Apache Samza, Apache Flink, Spring Cloud Data Flow, Amazon Kinesis, and Google Cloud Dataflow try to overcome the problems that traditional ETLs have. They are able to process large-scale, real-time or near real-time data, and perform data analysis efficiently. On the other hand, the use of various different frameworks brings problems along due to numerous Application Programming Interface (API), programming languages, and Software Development Kit (SDK). There are some software development kits to overcome that problem by unifying different streaming engines along. They are designed for helping developers to execute pipelines without mastery knowledge of the utilized implementation (Hesse et al., 2019).

Apache Spark is one of the most popular open-source and large-scale data processing frameworks for big data processing, analyzing, and querying. It was introduced to create enhanced and scalable ETLs and is a part of an effective ETL process by providing ease of use with different programming languages, different libraries for data processing, and data aggregation. Spark works in memory and is faster than traditional large-scale data processing frameworks. It provides client mode and cluster mode as deployment modes, and can connect to different cluster managers such as YARN, Kubernetes, Mesos,and Spark Standalone to allocate resources across applications. Cluster mode minimizes the network latency and is ideal for production cases while driver runs on a worker node. Therefore user does not have to be online during the job. On the other hand, client mode is frequently used for testing purposes and driver runs locally while tasks are running on worker nodes.

Apache Spark provides different libraries: Spark SQL for data processing, MLlib for machine learning implementations, GraphX for graph analysis, processing and computations, and Spark Streaming for real-time data processing. It supports the use of different data sources and programming languages such as Java, Scala, Python, and R (Karau & Warren, 2017).

Spark SQL module supports different data sources such as JavaScript Object Notation (JSON), Comma Separated Values (CSV), and text files, and has built-in support for structured streaming and data transforming. Another important feature of Spark SQL is supports different data abstractions and Domain-Specific Language (DSL) that provides APIs for Python, Java, Scala, and R. With the Spark SQL module, an ETL pipeline created on any language can get run in Apache Spark (Hesse et al., 2019).

2 Literature Review

2.1 General Overview

There is a variety of different Extract, Transform, and Load (ETL) tools in the market. They can be broken down into two main categories: Hand-Coded ETL Processes and Tool-Based ETL. Hand-Coded ETL processes are consisted of different programming languages to perform one ETL task. For instance, to extract data from a source, to transform data, to load the data to a target system; different programming languages such as Perl, SQL, and PL/SQL Bulk procedures could be used. Hand-Coded ETL tools require continuous modifications to integrate high-volume data that is produced from different sources while providing advantages in direct metadata creation and management, and easy unit testing. These tools are single-threaded and slow whereas modern ETL tools are multiple-threaded and faster in execution. On the other hand, Tool-Based ETLs are much preferable with additional functionalities and performance enhancement in terms of supporting multiple input or output databases, hyperdimensional design, diverse transforming functions, native databases or operating systems utility. They dispose of ETL workflow transformations, developing and maintaining complex routines. In addition, Tool-Based ETL tools provide user-friendly Graphical User Interface (GUI) and have features such as incremental aggregation, monitoring, and bulk loading (Kuchen et al., 2011; Pall & Khaira, 2018).

ETL tools also can be broken down into 4 sub-categories: Pure ETL Tools, Database Integrated Tools, Business Intelligence Tools, and Niche Products. There are commercial, open source, cloud-based, and custom ETL tools in the market such as Informatica, Talend, Google Cloud Dataflow, IBM, AWS Glue, Azure Data Factory, Oracle Data Integrator, SAS Data Management, Talend Open Studio, and Hadoop (Kuchen et al., 2011; Pall & Khaira, 2018).

Although traditional ETL tools are designed for and successful with batch processing, they do not function efficiently on near real-time data. Modern data management needs to cope with streaming and big data, real-time interactions, low latency, high security, and accuracy. Big data and real-time data analytics is a popular topic both in the industry and also in academia and brings new challenges affecting a wide range of industries. To extract value, give insight, and make sense of raw data, it is important to process the data. When it is a matter of big and real-time or near-real-time data, performance is a crucial metric due to the variability, velocity, and volume characteristics of data. Therefore, efficient big data stream processing frameworks are essential for processing large-scale data and designing, implementing, managing needed data pipelines and algorithms. There are various numbers of stream processing engines that are used both in industry and academia such as Apache Spark, Apache Storm, Apache Kafka, Apache Pulsar, Apache Samza, Apache Flink, Spring Cloud Data Flow, Amazon Kinesis, and Google Cloud Dataflow(Akidau et al., 2018).

For developers, having many frameworks poses some problems. The whole process comes up with different challenges due to the use of different programming languages, application programming interfaces (APIs), and software development kits (SDKs). Managing and handling all different and numerous technologies and systems with various APIs, writing and maintaining manyfold pipelines with different frameworks, and adopting a new framework bring difficulties along. Apache Beam offers a unified programming model for batch processing and streaming that can run across different distributed processing engines such as Apache Spark, Apache Flink, Apache Samza, and Google Cloud Dataflow. It helps developers to execute pipelines without further knowledge regarding utilized implementation. Pipelines are described with a single SDK and then they can get run by execution frameworks. In addition, frameworks can be exchanged without any code adapting processes(Hesse et al., 2019).

2.2 Apache Beam

Apache Beam is an open-source, unified analytics engine that provides SDKs in Java, Python, Go, SQL, TypeScript, and Scala languages and supports various execution engines such as Apache Spark, Apache Samza, Apache Flink, Apache Nemo, Google Cloud Dataflow, Hazelcast, and Twister2. It offers a unified programming model for both batch processing and also streaming processing. It also provides high-level abstraction and separates data from runtime and large-scale distributed data processing with cost-effectiveness, flexibility, portability, and extensibility (Li et al., 2018).

2.3 Apache Spark

In 2009, Matei Zaharia started to work on Apache Spark project during his Ph.D. at UC Berkeley (AWS Blog, n.d.). Apache Spark has introduced a new approach to data engineering and data science as an open-source, multi-language, in-memory, scalable, fault-tolerant, and distributed engine for processing data, implementing big data and machine learning algorithms, batch and graph processing, and performing real-time analysis for any size of data, but especially large-scale data on single-node machines or clusters (Gounaris et al., 2018). There are robust libraries that are built on the top of Spark: Spark SQL, GraphX, MLlib, and Spark Streaming. Apache Spark supports various programming languages such as Scala, Java, R, Python, and structured query language (SQL). It has become a popular framework both in industry and also in academia (Salloum et al., 2016)

Spark addresses the limitations of MapReduce in terms of data caching in RAM, reduced number of steps in a job, and reusing data across multiple parallel operations. Spark reuses data in-memory cache through Resilient Distributed Datasets (RDD), and abstractions of RDDs as DataFrames and Datasets (Belcastro et al., 2022). Owing to these features of Spark, it can be a hundred times faster than Apache Hadoop and it has quite lower latency than MapReduce (IBM Cloud Education, 2021). Apache Spark is essentially designed for batch-processing like ETL operations with high performance (Aziz et al., 2019; Salloum et al., 2016).

There are also some different projects and frameworks created in industry and academy on top of Apache Spark to support and boost the performance of ETL pipelines, perform efficient, distributed artificial intelligence and machine learning models while benefitting from Apache Spark's abilities (Machado et al., 2019; Santos et al., 2017; Suleykin & Panfilov, 2020; Widanage et al., 2020):
- Metadata Management Framework (MMF)
- Cylon
- DOD-ETL
- Lemonade

Within Apache Spark, Spark SQL module helps developers to query and transform data with the use of Structured Query Language (SQL), and it can be used as domain-specific language DSL. It provides high abstraction over Apache Spark core. It allows data extraction from different relational databases and diverse data file formats, applying transformations and queries with SQL, and loading data to target systems such as databases, data warehouses, BI tools, and different data files (Karau & Warren, 2017).

3 Requirements

This thesis aims to having an ETL data pipeline that is runnable and working on Apache Spark. The data pipeline should get the data from point A, transform it, and convey it to point B and run on Apache Spark. In order to achieve this goal, a data schema and data model are required to perform extract, transform and load processes. These components are received by choosing an open-source dataset, and the schema is defined for future uses in data extraction and reading stage, the first step of ETL data pipeline. The configured ETL pipeline must run on Apache Spark, complete data extraction from a data source, query and transform the data, and load it to a target system successfully. The ETL data pipeline should read and get the data which is in CSV format, transform and query, save it to an Excel file, and complete all these processes on Apache Spark. In addition, it is assumed that the initial configurations on Apache Spark concerning deployment mode, master machine, driver memory, driver cores, number of executors, executor memory, and executor cores are already done.

For creating an ETL data pipeline, a programming language must be selected to write the data pipeline script. Apache Spark can work with different programming languages: Java, Scala, Python, and R. In this study, Python programming language is chosen to perform extract, transform and load processes due to the fact that Python is flexible and easy to understand compared to other languages. Additionally, it provides advanced tools for data science, data mining, and data manipulation, and also it is popular and preferable among data analysts, scientists, and engineers. The PySpark interface enables writing Spark applications via Python API and provides a shell for interactive data analysis, and ensures collaboration between Spark and Python. Integrated Development Environment (IDE) can be chosen to write efficient scripts among different options such as PyCharm, Jupyter Notebook, Visual Studio Code, Spyder, and so on.

Other technical requirements in terms of tools and technologies to having a configured ETL pipeline on Apache Spark are having Java and Scala programming languages installed on the machine. Apache Spark can run on macOS, Linux, and Windows machines and currently requires Java 8/11/17, Scala 2.12/2.13, Python 3.7+, and R 3.5+ versions installed. The versions that are used in this study are Apache Spark 3.3.0, Scala: 2.12.15, Python 3.8.0, and Java: 11.0.2. (Spark, 2022).

Apache Spark	3.3.0
Python	3.8.0
Java	11.0.2
Scala	2.12.1

Table 3.1: List of Tools and Versions Used

Initial Spark configurations concerning deployment mode, master machine, driver memory, driver cores, number of executors, executor memory, and executor cores are considered already done at the beginning. This configuration options have a big impact on the performance of the data pipeline at the end. In this study, the aim is having an ETL data pipeline which can run on Apache Spark while performing data extraction, transformation and loading stages successfully.

4 Architecture

There are various numbers of stream processing engines that are used both in industry and academia such as Apache Spark, Apache Storm, Apache Kafka, Apache Pulsar, Apache Samza, Apache Flink, Spring Cloud Data Flow, Amazon Kinesis, and Google Cloud Dataflow. These different stream processing engines have different advantages and disadvantages (Kuchen et al., 2011).

Apache Spark is a popular, open-source distributed engine for large-scale data processing. Miscellaneous companies such as Amazon, eBay, and Alibaba use Apache Spark for performing quick data analysis to provide customized offers to improve their customer services (Belcastro et al., 2022). Spark is fault-tolerant and scalable, has fast runtimes performance, and supports multiple programming languages with different APIs. Therefore, the first step to configuring an ETL data pipeline on Apache Spark is choosing which programming language would best fit the implementation. The reason behind it is Apache Spark enables users to write pipeline scripts in different programming languages such as Scala, Java, Python, and R with its various APIs (Mukherjee & Kar, 2017)).

Every programming language has its pros and cons. Python is one of the most used programming languages nowadays because of its simple syntax, functionalities, and various libraries for data science, manipulation, and analysis. It is straightforward to learn, easy to write and run codes, and it supports writing codes with fewer lines practically. On the other hand, as Python is an interpreted language, the delivered performance is rather slower than compiled languages. The performance of Scala is much faster than Python. Another advantage of Scala is it also supports the usage of DataSets besides DataFrames, unlike Python. Yet, Scala is more difficult to learn and master as the learning curve is steeper. Java is wordier and more verbose than both Scala and Python (Salloum et al., 2016).

Spark SQL module supports structured data processing within Apache Spark programs and it is usable in Python, Scala, Java, and R programming languages. Spark SQL enables structured data querying by using SQL without any specific programming language knowledge and enables the use of DataFrame and DataSet abstractions. It also provides a common way to connect different data sources and data files. Catalyst Optimizer, the core of Spark SQL, optimizes structured queries plus DataFrames, and DataSets APIs and it is a powerful, and significant component of Apache Spark. In this thesis, the source data connection and four performed benchmark data queries are done with the Spark SQL module under the PySpark interface (Karau & Warren, 2017).

In conclusion, Python programming language is chosen to be used because of its simple syntax, ease of use, and advanced library support. Python and PySpark interface provide data reading, and saving to an Excel file (with an additional library of Python). Spark SQL module is used for data querying and processing, and initiating Spark session. It allows users to perform all data transforming stages by using SQL with DataFrame abstraction. Also, Spark SQL module is supported in all programming languages. The additional Python libraries that are used are openpyxl and Pandas which help to save the queried and processed data to the target file and keep the DataFrame format. Performance is not an important criterion in this study because of the small size of the used dataset which is 984 KB. (Mishra & Raman, 2019).

To sum up, the ETL data pipeline script is written in Python by using PySpark interface and Spark SQL module. In general structure, Spark SQL is used with Structured Query Language to extract and transform the data. Pandas and openpyxl libraries are used at the end to load data into an Excel file by saving the current DataFrame format. For running the ETL data pipeline on Apache Spark, the Python code file is sent by spark-submit command by creating a Standalone Cluster. The performance of the data pipeline is measured by using Docker Container and Spark GUI.

5 Design and Implementation

This chapter is divided to 2 sections for explaining ETL data pipeline configuration structure more clearly.

5.1 Building an Apache Spark Session with PySpark

There are different ways to launch a Spark application. Spark-submit command on terminal or command window or PySpark shell can be used to initialize and run ETL data pipeline scripts on Apache Spark. Master, deployment mode, driver memory, executor memory, number of executor cores configurations can be set on shell. After submitting the Python file including ETL data pipeline script on the shell, the whole process can be tracked on Spark Web UI.

Deployment mode selection can be set considering whether the purpose is debugging or production. Client mode is suitable for debugging and testing scenarios while cluster mode fits best to production cases. There are different kinds of cluster managers such as Spark Standalone, Kubernetes, YARN, and Mesos. Spark can also run locally with one worker thread. In addition, memory allocations can also be set based on the sources in use. In this thesis, it is assumed that initial Spark configurations concerning the options above are already done and met the requirements of use scenario.

In order to have an ETL data pipeline that can run on Apache Spark, the Spark session has to be built within the script. SparkSession helps to run queries, cache tables, and read files such as CSV, JSON, and Parquet. More options and configurations can also be passed to SparkSession under the builder class. The builder class assists to create a Spark session. App name is optional and used to name the session which will be visible through the Spark Web user interface. Get or create checks if there is an already existing session on Spark. If there is not any existing session, it creates a brand new one with the configurations passed

through the builder. Once Spark session is initialized, the dataset can be read, transformed, and queried. If the application will be deployed in cluster mode, additional options such as master, driver memory, driver cores, executor memory, executor cores, and so on can be added to the script.

```
from pyspark.sql import SparkSession

spark = SparkSession \
        .builder \
        .appName("App Name") \
        .getOrCreate()
```

Listing 5.1: Creating a Spark Session on Local Machine

For a better performance and sophisticated configuration options, different cluster managers can be used and further configurations can be made.

For running the ETL data pipeline script on Apache Spark, different IDEs can be chosen based on personal preferences. Jupyter Notebook helps to see every line's output interactively. PyCharm Professional Edition provides big data tools plugins including Apache Spark, therefore it can run the Python core and the ETL pipeline script directly on Apache Spark without spark submitting on terminal, command window, or Spark shell and may save time based on the use scenario. On the other hand, it is also possible to run the script directly on Spark shell or on terminal via spark-submit command.

5.2 ETL Pipeline Processes

There are three main API types in Apache Spark with different abstraction levels to processing large volumes of data: RDD, DataFrame, Dataset. RDDs (Resilient Distributed Datasets) are immutable and distributed collections of Python and Java objects that are divided across a cluster. They have different functions such as map, join, flatMap, reduce, and filter which can be applied through the whole dataset. The functions are sent to the nodes on the cluster to manipulate and process distributed datasets or perform input and output functions to read and write the dataset between Spark Java Virtual Machines and distributed storage system. RDDs are evaluated by Spark lazily. RDD transformations are computed solely and exclusively in case of final RDD data needs to be computed. The lazy evaluation approach and immutability increase efficiency, scalability, ease of use, and fault tolerance and reduce computational complexity (Karau & Warren, 2017; Mishra & Raman, 2019).

RDD API best fits for low-level abstractions and low-level operations, particularly with unstructured data. It can not benefit from Spark's Catalyst optimizer and Tungsten in-memory coding engine and is not able to infer the schema of structured data. The other 2 APIs in Spark, DataFrames and Datasets, are more convenient for structured and semi-structured data. These 2 APIs work on top of Spark SQL by using Catalyst optimizer and Tungsten fast encoding (Mukherjee & Kar, 2017). Both DataFrames and Datasets have a sufficient performance on high-level abstractions, domain-specific APIs, and high-level expression demanding applications including filterings, aggregations, complex SQL queries, mappings, columnar access, and lambda functions. For taking the advantage of Catalyst optimization and Tungsten's efficient code generation, and higher type safety at compile time, Dataset API would be the most suitable option. Dataset API works in Java and Scala programming languages equally while DataFrame is much more Scala-centric. Besides, considering Python and R programming languages do not have runtime type safety, DataFrame API should be used if the user is going to write Apache Spark applications with these two programming languages due to the lack of Dataset API support. (Salloum et al., 2016).

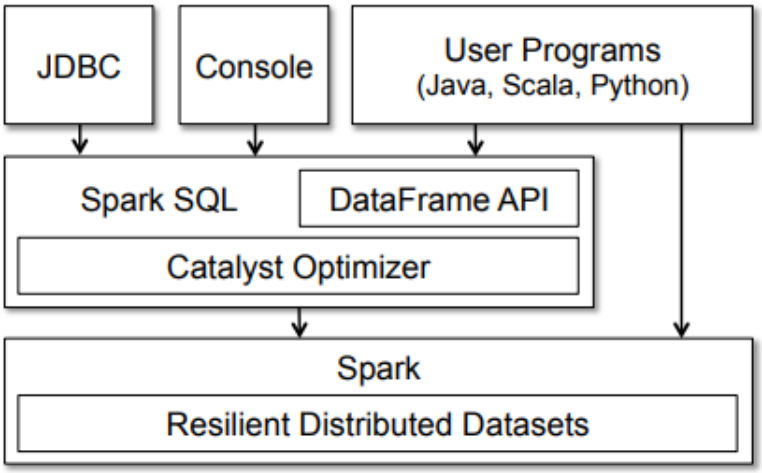

Figure 5.1: Spark SQL Architecture (Sarkar, 2019).

In this study it is aimed to obtain an ETL data pipeline that can get the data from point A (source system), transform it, and convey it to point B (target system) and is runnable on Apache Spark. To achieve this aim, it is planned to use the Python programming language and the PySpark interface.

The source data has a structured schema and it is stored in a CSV file which has a size of 984 KB. DataFrame API within Spark SQL module is used to perform further operations on data. For query-running and data manipulation, PySpark has to be downloaded and additional modules should be imported. Within the PySpark interface, Spark SQL module can be used for data querying and transforming processes. Additionally, Pandas and openpyxl libraries are used for saving the transformed data to an excel file in DataFrame format.

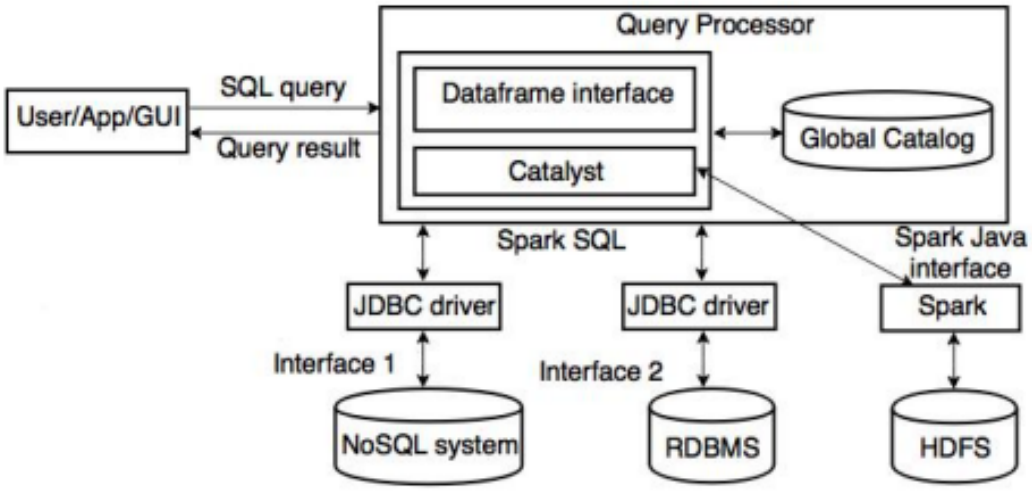

Figure 5.2: Spark SQL Architecture (Bondiombouy & Valduriez, 2016).

The dataset that is used in this study has a structured schema and consists of ten columns: Row ID, Order ID, Order Date, Ship Mode, Customer ID, Postal Code, ProductID, Category, SubCategory, and Sales. The source format is a CSV file with 984 KB size. While reading the CSV or any other data file format with Spark SQL module, different parameters such as header, inferSchema, and delimiter can be used beside the file path. InferSchema helps to define the current data structure while reading a file. In case of inferSchema is disabled, Spark does not analyze the data schema. In this scenario, a valid schema must be declared with StructType object containing different StructFields within. StructType provides the flexibility for developers to define the data schema by themselves.

In order to query and transform data, 4 benchmark queries are selected and applied to dataset. For all steps including data reading, transforming and querying, SparkSQL module is used within PySpark interface. In the end, transformed data can be load to any target system. The data is saved as an Excel file including 4 separate pages, each having the results of four different queries. By submitting the ETL pipeline Python script to Apache Spark, the aim of having a configured ETL data pipeline that runs on Apache Spark is acquired.

Applied queries include currency conversion, grouping and selecting different attributes, and calculating total and average sales amount.

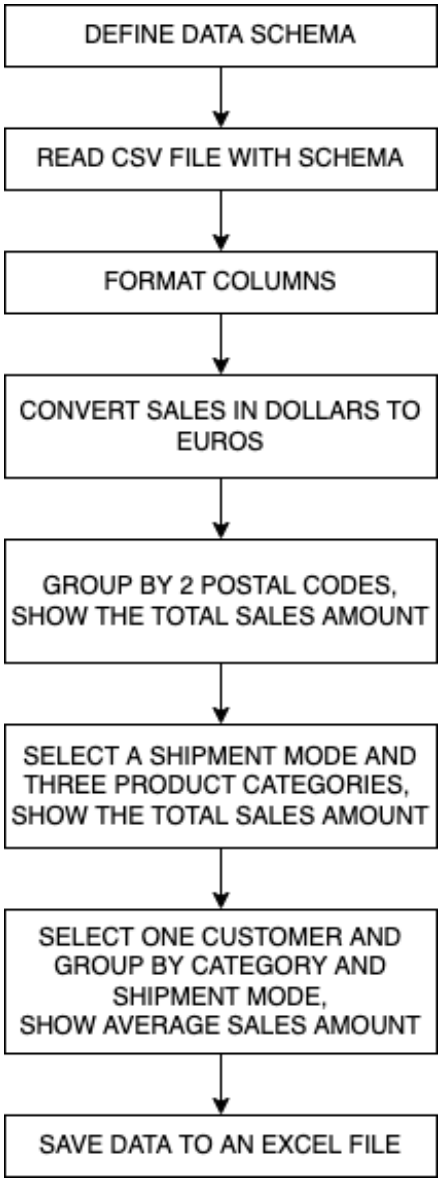

Figure 5.3: Extract, Transform, and Load Flow

In the data schema defining stage, InferSchema parameter can be set to true which is default or it can be manually defined by the developer to have flexibility in special use cases.

Field Name	Data Type	Nullable?
Row ID	Integer	Not
Order ID	String	Not
Ship Mode	String	Not
.		
.		

Table 5.1: Data Schema

```
schema = StructType([StructField("Row ID", IntegerType(),False),
                     StructField("Order ID",StringType(),False),
                     StructField("Ship Mode",StringType(),False)
                     ..........])
```

Listing 5.2: Schema Definition in Python

Spark SQL is capable of extracting data with defined schema and reading it directly.

```
sparkData = spark.read.schema(schema).option("sep",",").\
                option("header", "true").\
                csv(r"<DATA PATH>")
```

Listing 5.3: Data Extraction

Spark SQL module helps to run queries with SQL as a domain-specific language. It unifies data querying in applications that are using different programming languages.

```

df2 = sparkData.select("....").\
            where(sparkData["Ship Mode"] == "1st Class").\
            where((sparkData["Sub-Category"] == "Binders")|
                 (sparkData["Sub-Category"] == "Paper")|
                 (sparkData["Sub-Category"] == "Labels"))
```

Listing 5.4: Example Query

Spark SQL module does not directly support to save data in an Excel file. Therefore, pandas library, which is quite popular among people working with Python programming language, is used to save DataFrame structure of the data. In addition, openpyxl library is used as an engine to save data in an Excel file. At the end, every result of four benchmark queries are saved to different Excel sheets by using pandas and openpyxl libraries.

```
1 with pd.ExcelWriter("<FILE PATH>",mode="w", engine='openpyxl') as
      writer:
2
3     df0.toPandas().to_excel(writer, sheet_name="query1", index=
     False)
4     .
5     .
6     .
```

Listing 5.5: Data Load

6 Evaluation

The evaluation criteria for this thesis are as follows:

Criteria 1: Benchmark queries must be applied successfully to the dataset.
✓**This requirement has been satisfied,** four Benchmark queries are applied to the dataset. More queries would have been applied but as static data which is stored in a CSV file is used in this thesis, the benchmark queries requiring real-time data could not be involved and applied.

Criteria 2: Queried data has to be loaded to a target system (database, data warehouse, or file).
✓**This requirement has been satisfied,** queried and transformed data is saved to an Excel file on four separate pages.

Criteria 3: Data pipeline that is written in determined programming language shall be runnable on Apache Spark.
✓**This requirement has been satisfied,** the ETL pipeline that is written in Python programming language can run on Apache Spark.

All required criteria are met. Yet, as a nice-to-have, change data capture (CDC) would have been implemented to use system resources more efficiently and improve data integration. CDC technique may be considered to use in in further studies.

An ETL data pipeline that extracts a CSV file with a size of 984 KB, reads data with a pre-determined data schema, transforms data by converting currency, selecting attributes and grouping, and doing summation and averaging is created on Apache Spark. As the last step, ETL data pipeline loads all transformed data into an Excel file called "output" which involves 4 different pages including 4 different data query results on each page.

The tracked performance metrics regarding the ETL data pipeline are processing time and CPU utilization (%). To measure the CPU utilization of the ETL data pipeline which is runnable on Apache Spark, Docker container is used and and the average CPU utilization is measured as 21.10% during the whole application. Time spent to run the ETL data pipeline is measured as 17 seconds as shown on Spark GUI.

7 Conclusion

As Big Data is one of the most popular phenomena nowadays, the performance of data extraction and transformation, machine learning model training, and data analysis topics have gained more importance than ever. When it is a matter of high variety, volume, and velocity of the data, big data frameworks would be more helpful than the traditional ETL approaches to process data due to the scalability, additional library support, and data streaming abilities. There are various and different streaming engines. Apache Spark is one of the most effective and demanding tools and provides ease of use to create scalable ETLs. It provides useful libraries for data processing and APIs for working with different programming languages. This thesis aims to propose a method to compile a data pipeline that has a valid schema on Apache Spark as the stream processor. The approach used in this thesis is using Structured Query Language (SQL) with Python programming language for data extraction, transform, and load which is supported by Spark SQL module and PySpark interface.[1]

Apache Spark provides support for different programming languages yet as Python is an already interpreted language and slower than compiled ones, even though Spark provides support and API with PySpark, it costs performance issues. For having the best performance result, Scala would be a better choice to run Spark applications. Also, initial configurations to achieve a better performance are quite complicated and requires manual optimization calculations. Another problem with working with PySpark is that there is not as much available information as Scala implementations have. And another problem that is encountered is that some libraries are not supported anymore to load and write data to a file by PySpark interface. The performance metrics of the ETL data pipeline, CPU Utilization % and time spent, are measured via Docker container and Spark UI.

[1] https://github.com/gbatmaci/SparkETL

References

Akidau, T., Chernyak, S., & Lax, R. (2018). *Streaming systems: The what, where, when, and how of large-scale data processing*. O'Reilly Media. https://books.google.de/books?hl=tr&lr=&id=TAxlDwAAQBAJ&oi=fnd&pg=PT8&dq=akidau+2016&ots=p7zsyiPT4Z&sig=lfyMVZ2t3BhzQgMXqXTfhLxXHNk#v=onepage&q=akidau%5C%5C%202016%5C%5C&f=false

Aziz, K., Zaidouni, D., & Bellafkih, M. (2019). Leveraging resource management for efficient performance of apache spark. *Journal of Big Data*, *6*(1), 78. https://doi.org/10.1186/s40537-019-0240-1

Belcastro, L., Cantini, R., Marozzo, F., Orsino, A., Talia, D., & Trunfio, P. (2022). Programming big data analysis: Principles and solutions. *Journal of Big Data*, *9*(1), 4. https://doi.org/10.1186/s40537-021-00555-2

Bondiombouy, C., & Valduriez, P. (2016). Query processing in multistore systems: An overview. *International Journal of Data Science and Analytics*, *5*(4), 309. https://doi.org/10.1504/IJCC.2016.080903

Hesse, G., Matthies, C., Glass, K., Huegle, J., & Uflacker, M. (2019). *Quantitative impact evaluation of an abstraction layer for data stream processing systems*. IEEE. https://doi.org/10.1109/ICDCS.2019.00138

Karau, H., & Warren, R. (2017). *High performance spark* (Vol. 1). O'Reilly Media. https://doi.org/10.1007/s41060-016-0027-9

Kuchen, H., Jansen, T., & Majchrzak, T. A. (2011). *Efficiency evaluation of open source etl tools*. ACM Press.

Li, S., Gerver, P., MacMillan, J., Debrunner, D., Marshall, W., & Wu, K.-L. (2018). Challenges and experiences in building an efficient apache beam runner for ibm streams. *11*, 1742–1754. https://doi.org/10.14778/3229863.3229864

Machado, G. V., Cunha, Í., Pereira, A. C. M., & Oliveira, L. B. (2019). Dod-etl: Distributed on-demand etl for near real-time business intelligence. *International Journal of Data Science and Analytics*, *10*(1), 21. https://doi.org/10.1186/s13174-019-0121-z

Mishra, R. K., & Raman, S. R. (2019). *Pyspark sql recipes: With hiveql, dataframe and graphframes*. Apress. http://link.springer.com/10.1007/978-1-4842-4335-0

Mukherjee, R., & Kar, P. (2017). *A comparative review of data warehousing etl tools with new trends and industry insight*. IEEE. https://doi.org/10.1109/IACC.2017.0192

Pall, A. S., & Khaira, J. S. (2018). *A comparative review of extraction, transformation and loading tools*. https://www.researchgate.net/publication/350132800

Raghuraman, M. T., & College, G. N. (2021). A assessment of etl tools, 1–5. https://www.researchgate.net/publication/350132800

Salloum, S., Dautov, R., Chen, X., Peng, P. X., & Huang, J. Z. (2016). Big data analytics on apache spark. *International Journal of Data Science and Analytics*, *1*(8), 145–164. https://doi.org/10.1007/s41060-016-0027-9

Santos, W. d., Carvalho, L. F. M., Avelar, G. d. P., Silva, A., Ponce, L. M., Guedes, D., & Meira, W. (2017). *Lemonade: A scalable and efficient spark-based platform for data analytics*. IEEE. https://doi.org/10.1109/CCGRID.2017.142

Sarkar, D. (2019). *Scaling relational databases with Apache Spark SQL and DataFrames architecture and features*. Retrieved August 31, 2022, from https://opensource.com/article/19/3/sql-scale-apache-spark-sql-and-dataframes

Spark, A. (2022). *Spark Overview downloading*. Retrieved August 3, 2022, from https://spark.apache.org/docs/latest/

Suleykin, A., & Panfilov, P. (2020). *Metadata-driven industrial-grade etl system*. IEEE. https://doi.org/10.1109/BigData50022.2020.9378367

Widanage, C., Perera, N., Abeykoon, V., Kamburugamuve, S., Kanewala, T. A., Maithree, H., Wickramasinghe, P., Uyar, A., Gunduz, G., & Fox, G. (2020). *High performance data engineering everywhere*. IEEE. https://doi.org/10.1109/SMDS49396.2020.00022