Selection, Modelling and Evaluation of Workflow Management Systems

Data Engineering Master's Thesis

30005931

submitted by

Can Yetismis

with supervision from

Dr. Stefan Kettemann and

Dr. Christian A. Müller

Constructor University

May 2023

Statutory Declaration of Authorship

English-Declaration of Authorship: I hereby declare that the thesis submitted was created and written solely by myself without any external support. Any sources, direct or indirect, are marked as such. I am aware of the fact that the contents of the thesis in digital form may be revised with regard to usage of unauthorized aid as well as whether the whole or parts of it may be identified as plagiarism. I do agree my work to be entered into a database for it to be compared with existing sources, where it will remain in order to enable further comparisons with future theses. This does not grant any rights of reproduction and usage, however. This document was neither presented to any other examination board nor has it been published.

German-Erklärung der Autorenschaft (Urheberschaft): Ich erkläre hiermit, dass die vorliegende Arbeit ohne fremde Hilfe ausschließlich von mir erstellt und geschrieben worden ist. Jedwede verwendeten Quellen, direkter oder indirekter Art, sind als solche kenntlich gemacht worden. Mir ist die Tatsache bewusst, dass der Inhalt der Thesis in digitaler Form geprüft werden kann im Hinblick darauf, ob es sich ganz oder in Teilen um ein Plagiat handelt. Ich bin damit einverstanden, dass meine Arbeit in einer Datenbank eingegeben werden kann, um mit bereits bestehenden Quellen verglichen zu werden und dort auch verbleibt, um mit zukünftigen Arbeiten verglichen werden zu können. Dies berechtigt jedoch nicht zur Verwendung oder Vervielfältigung. Diese Arbeit wurde noch keiner anderen Prüfungsbehörde vorgelegt noch wurde sie bisher veröffentlicht.

Family Name, First Name	Yetismis Can
Matriculation Number	30005931
Thesis Type	Masters Thesis
Date	15.05.2023
Signature	

Abstract

The ever-growing big data can provide humanity valuable information when used in different application cases. Although, big data itself needs to be processed before an insight could be drawn out of it. There are many different stages involving the processing of big data, and the management of these processes are a challenge of their own. Hence data pipelines were introduced in order to manage these complex processes. However the structure and orchestration of the pipelines could easily get out of control when a multitude of tasks are introduced. As a result, workflow management systems were created to manage pipelines. There are many different implementations for workflow management systems and the literature surrounding workflow management is immense. Though the resources are plentiful, selection in between workflow management systems are a considerable challenge and most of the resources does not discuss the selection in detail. This thesis is a research attempt to select a workflow management system, create a pipeline model looking at a real-life use case, and finally evaluate the model using a set of criteria to determine if the tool is efficient for the provided use case.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The amount of data created is ever increasing in each passing day. A 2021 survey from Statista projects this increase to be around 181 zettabytes (a zettabyte is 1 billion terabytes [1]) in size by the year 2025 [2]. A more recent study states that people generate 2.5 Quantillion bytes of data every single day [3]. Naturally, the data can assumed to keep growing in each passing day.

Such vast quantities of data is generally referred as *Big Data* [4]. Big Data has many applications, to list a few, examples may include: health care, manufacturing, IoT and media and entertainment [4]. Therefore it is possible to say that when utilised, big data may provide a lot of insights depending on the case it is being used at [5].

1.1 How to Utilise Big Data?

Big data does not have a use unless it has a business value [5]. Hence many establishments analyse big data to make sense and draw conclusions [5]. However, analysis of big data is complex challenge [5]. The term does not only refer to the size of the data, it also refers to the complexity of the structure of the data, as well as the arrival of data in huge velocities [5]. Hence the analysis of big data tend to contain many different stages [6]. A brief overview is like in the following [6]:

- Analysing what kind of data is required
- Preparation and selection of the data
- Data Pre-Processing and data cleaning
- Transformation of data for the interpretation by analytic tools
- Analysis of the data

- Evaluation of the findings

Due to many complex stages being involved, it is natural to assume that the orchestration and processing of the tasks is a considerable challenge, especially considering pipelines that contain a multitude of tasks. Hence it is crucial to manage the aforementioned processing effectively.

1.2 Data Pipelining and Orchestration

Such complex processes are managed through so-called data pipelines, which is an abstract term given to an environment that is managing a set of complex chains of actions [7]. Depending on the implementation, pipelines provide many advantages such as automated data management, monitoring or fault detection [7]. However modelling and implementation of such an environment from ground-up pose challenges in its own right and could be an entire software project by itself. There are existing tools in the industry to ease the deployment time of the pipelines. In a general sense these pipeline orchestration tools are referred as workflow management systems [8, 9]. However they have different design principles as well as different dependencies for becoming operational [10]. Hence selection of workflow management systems is a challenge by it's own.

1.3 Research Question

Considering the aforementioned challenges, the research question for this thesis has determined to be the following:

How to select, model and evaluate an efficient Workflow Management System to provide a data pipeline for internally dependent tasks?

As per Chapter 3, the literature surrounding workflow is quite immense and ever-increasing in each passing day. Although the literature is plenty, most of the resources does not provide a comprehensive review regarding the selection of the workflow management systems. Hence the main goal of this research is to provide an approach to select and evaluate a workflow management system to see if the tool could be considered as efficient or not. For evaluation purposes a real life use case has been selected, in which two pipelining approaches were modelled and an evaluation has been performed with respect to the criteria determined for efficiency.

Chapter 2

Background

This chapter is to explain some of the most commonly used terminology, tools and concepts within the context of the thesis. The core idea of this chapter is to define what an "efficient" workflow management system is. To do so it is crucial to define some software related terms to lay a foundation on the discussion.

2.1 Directed Acyclic Graphs

A graph is a mathematical diagram consisting of two abstract components: vertices and edges [11]. Graphs are typically used to represent real-life situation. The projection is achieved through a crude diagram consisting of points and lines, which are called vertices and edges respectively [11]. To visualise the concept better, a real-world situation and an example for it's graph could be observed on the Figure 2.1:

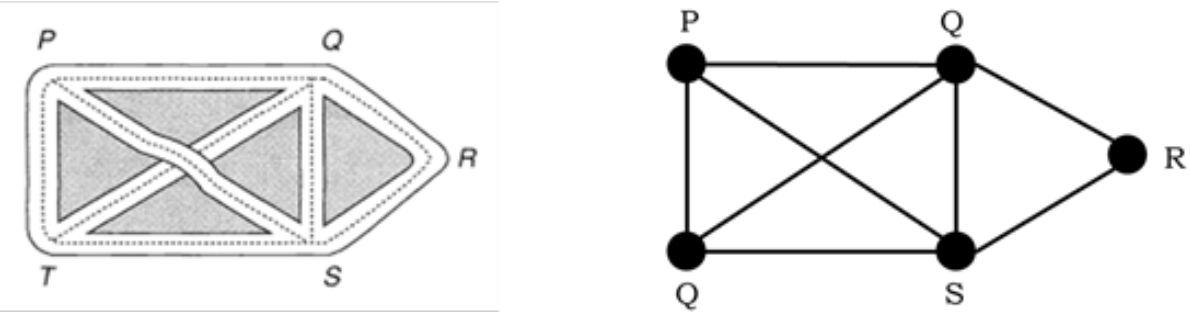


Figure 2.1: A part of a road map (left) versus it's graph representation (right). Figure on the left is taken from [11] and the figure on the right is reconstructed based on the example provided by [11].

Edges on mathematical graphs could also contain directions, making the graphs directed [11, 12][1]. On directed graphs, if a walk (a movement from one vertex to another)

[1] Resources [11, 12] may refer to directed graphs as *digraphs*

ends on the same vertex where the traversal started from, then the graph contains a cycle, hence making the graph cyclic [11, 12]. If no such walk is possible then the graph becomes acyclic. A comparison in between cyclic directed graph and acyclic graphs can be observed on Figure 2.2:

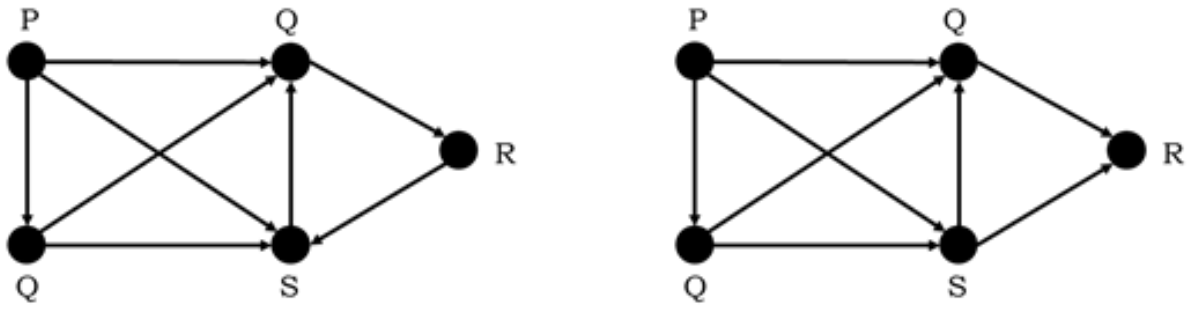


Figure 2.2: A cyclic (left) and an acyclic (right) graph, derived from graph in Figure 2.1.

As can be observed, a walk such as $Q \rightarrow R \rightarrow S \rightarrow Q$ is possible on the graph on the left, whereas no such traversal is possible for the graph on the right. Hence, by mathematical definition the graph on the left is a directed cyclic graph, and the graph on the right is a **directed acyclic graph** or **DAG** for short.

2.2 Pipelines and Workflow Management Systems

In the context of data, a **pipeline** can be described as a set of processes for moving and transforming data from a variety of sources to a destination or destinations [13]. Pipelines, being an essential part of the *Data Engineering* discipline are crucial for analytics, reporting and machine learning processes [13]. Pipelines also tend to have a variation in complexity depending on the application. [13].

A **Workflow Management System**[2] or **WMS** for short is a tool for scheduling and controlling the flow of tasks within the data pipeline [8, 9]. Some readily available WMSs are more generic by design such as Apache Airflow or Spotify Luigi whereas others focus on specific platforms to operate such as Kubeflow [8]. A common way to represent pipelines for some WMSs is through DAGs [8,9]. DAGs ensure that the tasks are executed on the order they are defined, providing a guarantee that no task is executed before their dependencies/upstreams are done executing [8, 9].

[2]Resources [8, 9] refer to WMSs as *Workflow orchestration platforms* or *Pipeline orchestration tools*. Both of these names refer to the same concept.

2.3 Scalability

In a general sense, **scalability** refers to a program's or a system's ability to handle the increasing work-load [14–16]. Scalability is an widely accepted concept within the discipline of *Software Engineering* and there are no unified procedures in literature for measuring it [16]. However, with respect to the previous definition some other software concepts may provide an insight into whether a program or a system is scalable or not.

The terms related to scalability will be discussed within two contexts: Scalability of software and scalability of distributed systems (distributed networks of computational nodes). Figure 2.3 illustrates the breakdown of the term scalability.

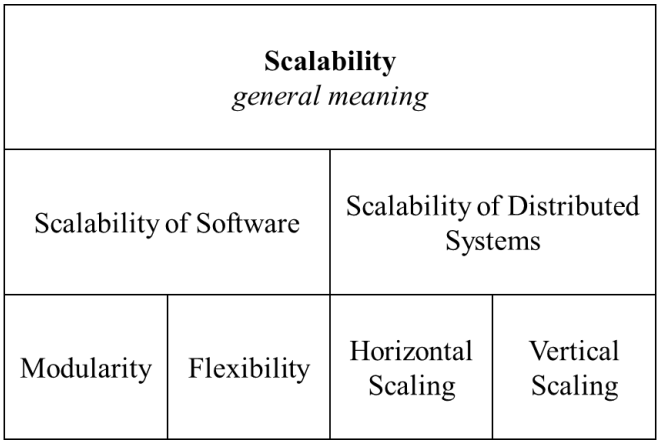


Figure 2.3: Breakdown of the term scalability. The top layer represents the general definition of scalability in software engineering discipline, middle layer represents a further categorisation of scalability with respect to the contexts of software and distributed systems. Bottom layer represents the software concepts that help to explain the contexts.

An insight for the **scalability of software** could be provided with the following concepts:

- **Modularity:** the degree that the source code can be divided up to smaller re-usable modules or components [17, 18]. Modularity can allow parts of software to be executed concurrently or parallel, which in return gives ability to the program to scale up depending on the work-load.
- **Flexibility:** the degree that a software can be adapted or changed with respect to the demands of the environment that it is deployed at [17, 19]. A flexible software can incorporate more components within it's structure on demand, to handle required changes for balancing the work-load.

Similarly **scalability of distributed systems** could be determined with the following concepts:

- **Vertical Scaling:** scaling achieved through providing more system resources to a single computational node, such as increasing the number of CPU cores or adding more RAM [20].
- **Horizontal Scaling:** scaling achieved through adding more identical computational nodes within a distributed system and sharing/balancing the work-load in between [17, 20]. It is generally preferred over vertical scaling when large data flows need to be processed [20].

A graphical illustration for vertical and horizontal scaling can be observed on Figure 2.4:

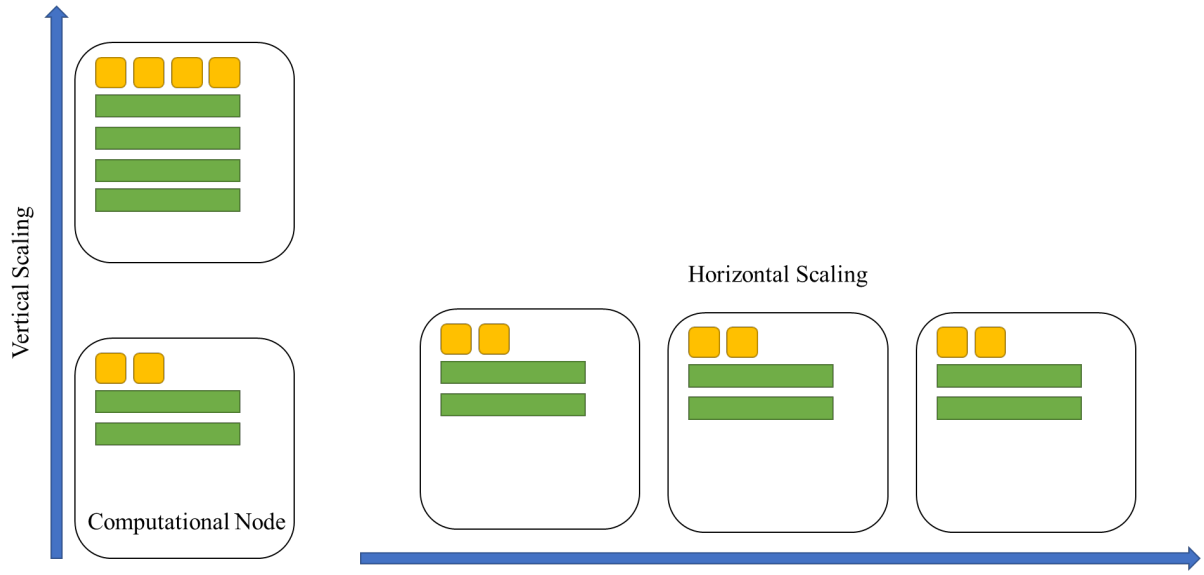


Figure 2.4: Vertical versus horizontal scaling of a computational node. Yellow boxes represent CPU cores and green rectangles represent RAM sticks. The bottom left node represents the starting point for the computational node in this figure, which is a single computer with two cores and two ram sticks.

2.4 Monitoring

Monitoring can be defined as the process of surveilling a system's or a software's state changes and data flow, to ensure that the system or the software is behaving in a desired way [21]. Monitoring is effective for observing the points of failure, hence contributing greatly for the detection of errors or the points of failure [21]. Some WMSs, such as AirFlow, also provide a means of monitoring within their architecture (see Chapter 6.4).

2.5 "Efficient" Workflow Management System

Before discussing efficiency in the context of WMSs, it is important to define the word efficiency within the English language. Cambridge Dictionary defines efficiency as *"working or operating quickly and effectively in an organized way"* [22]. Hence, it is also important to define the word effectively, and Cambridge Dictionary defines the word effectively as *"in a way that is successful and achieves what you want"* [23].

With respect to the definitions made on this section, and the definition provided in Section 2.2 a definition for an efficient WMS can be made. An **efficient WMS** is a WMS that can process and move data through the various stages of the pipeline in a swift and effective manner. An insight on swiftness and effectiveness can be provided by considering following concepts:

- **Short processing time:** A WMS having a short processing time for a pipeline means that more pipelines can be executed at a given time interval. Hence more data-related operation can be completed.
- **High data throughput:** Throughput refers to the amount of data processed within a given time interval [17]. Higher throughput means more data is being processed at a given time.
- **Scalable:** An efficient WMS should be able to counter any increase in workload without interrupting the data flow. Hence scalability is also crucial

Although not directly related to the efficiency, it is also important to discuss two more concepts that would be an expected characteristic for an efficient WMS. First is the ability to provide the desired data output. This criterion is highly correlated with the implementation of the pipelines themselves as well as the design of the WMS itself. Hence it will not be a used for evaluation purposes. The second criterion is the availability of monitoring, which greatly impacts the deployment and maintenance of pipeline implementations. Although monitoring is independent from efficiency, it will be taken in consideration while evaluating.

Chapter 3

Problem Statement

This chapter will discuss why there is a need for workflow management systems, existing literature for pipelining and workflow management systems and workflow management system selection, and finally an example use-case where a workflow management system can be utilised.

3.1 Why do Workflow Management Systems Exist?

Haines [24] state that the primary reason relates to addressing of two commonly faced problems when dealing with data:

- **Bad data Upstreams:** When constructing a pipeline from the scratch, it is common to see problems such as some sections of the pipelines failing, affecting the delivery of data.
- **Time:** In many use cases, data has to be delivered in a desired schema or state, within a defined time interval[1].

WMSs exist to solve the aforementioned issues by providing environments to create "blueprints" to orchestrate the data-flow in an efficient manner (see Section 2.5) [24].

Another important aspect that can be discussed is most of the WMSs considering **scalability aspects** (see Section 2.3). Harenslak and de Ruiter [10] argues that most of the existing WMSs utilise a method of definition of tasks in separate modules, that can be added to DAGs over time, which provides modularity and flexibility. Harenslak and de Ruiter [10][2] also argues some WMSs consider horizontal-scalability and parallel-execution through the utilisation of distributed-computing platforms such as Kubernetes (see Section REF HERE) [25].

[1] [24] specifically mentions time aspect within a business context.

[2] [10] use the word *Anywhere* for *Installation Platforms* column on the *Table 1.1* in p. 9, within the context the terms also refer to Kubernetes

3.2 Literature Review and WMS Selection

Due to the big data becoming more and more prevalent nowadays, the literature on data pipelines and WMSs also started to increase. There has been an emergence of books recently, written on the subject of pipelining and WMSs. Examples may include *Data Pipelines Pocket Reference* by James Densmore, *Building Machine Learning Pipelines* by Hannes Hapke and Catherine Nelson, or *Data Pipelines with Apache Airflow* by Bas P. Harenslak and Julian de Ruiter. Densmore talks about the construction of data pipelines and available tools that may help with the construction [13][3]. Hapke and Nelson focus more on the data pipeline construction within a machine learning environment, as well as WMSs that may automatise the pipeline executions [9]. Harenslak and de Ruiter discuss the implementation of pipelines using Apache Airflow [10].

The subject also sparked an interest within the scientific and business communities, yielding in an excess of various theses and papers surrounding the topic. Examples vary from applications for a single use case to review of the subject in depth. Finnigan and Toner [26] describe an Airflow implementation to harvest and store digital resources. Hilgendorf [27] talks about an industrial Airflow implementation for processing vehicle data obtained through IoT devices, as well as the reasons to choose Airflow. Mitchell et al. [28] reviewed four WMSs to discuss their application cases as well as their strengths and weaknesses. Mitchell et al. also proposed some criteria to justify how to select a WMS.

Although the aforementioned resources provide a detailed insight on application cases for the data pipelines and WMSs, the discussion regarding the the selection is limited. As per the WMS description made on Section 2.5, selection of a WMS can be considered as an optimisation problem. An **optimisation problem** is a computational problem where the goal is to find the best solution out of all the existing solutions [17, 29]. Therefore, it is possible to say that a careful consideration is needed to make such a selection [10].

A master's thesis from Panu Kortelainen [30] argues that the choices that can be made for the WMSs are vast, and the variation in between the project and technical characteristics WMSs is high, making the choice a considerable challenge [30]. As a solution Kortelainen lays a detailed criteria for the selection of WMSs depending on the desired application case. Hence his work will form a basis for the WMS selection for this thesis.

[3]The citations in this section is to refer to the books themselves, rather than the stated page numbers within the citations.

3.3 Case Study on WasteAnt

WasteAnt is a company that provide solutions on performing a variety of automated assessments on waste streams, through the utilisation of machine learning and artificial intelligence techniques [31]. Company gathers a variety of data through the sensor-boxes installed on waste incineration plants and processes them within the company's image processing back-end system[4]. A brief overview of the workflow of WasteAnt back-end can be observed on Figure 3.1:

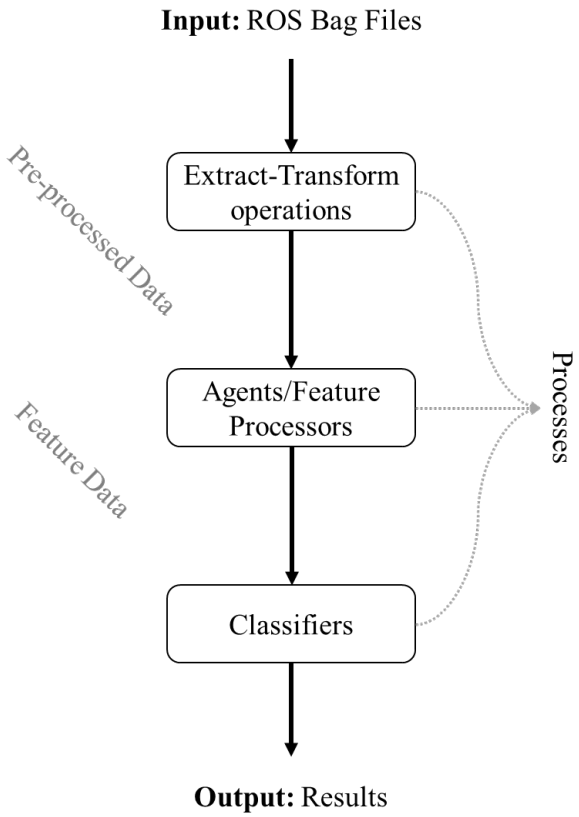


Figure 3.1: Workflow of processes (boxes) and data transformations in company's back-end infrastructure.

First the sensor data arrives to the company's back-end in the form of Robot Operating System (ROS) bag files, which is a ROS specific file format for storing sensor data [32]. After sensor data is obtained, the data is fed into a series of extract-transform tasks that pre-process the sensor data. Once data is-pre processed, it is then fed into components referred as agents or feature processors. Agents/Feature Processors are software components that apply a variety of image processing approaches to extract features out of pre-processed sensor data. Then the extracted features are fed into the last component

[4]The details regarding company infrastructure were obtained through interviews with the founders of WasteAnt: Dr. Christian A. Müller, Dr. Arturo Gomez Chavez and Dr. Szymon Krupinski.

of the pipeline which are classifiers. Classifiers perform classification on the feature obtained from agents and generate a result that is then sent back to the incineration plant operators.

Company handles pipelining through the utilisation of ROS's inbuilt publisher-subscriber (pub-sub) system [33]. The downstream tasks publish messages regarding their executions and upstream processes subscribe to the upstream's publications. A brief overview of the concept can be observed on the Figure 3.2:

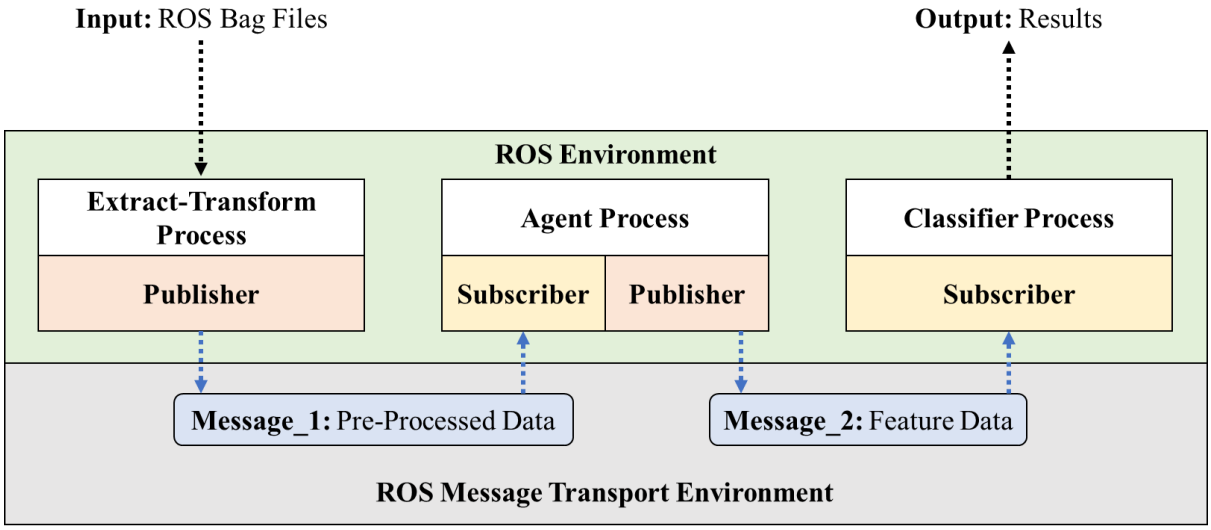


Figure 3.2: A singular pipeline within WasteAnt back-end. Figure demonstrates the interaction of processes with the ROS Environment [34, 35] and ROS Message Transport Environment [34–36]. The actual infrastructure contains multitude of processes. Classifier and agent nodes may have multiple dependencies, which vary from process to process.

The approach brings limitations in parallel. First problem arises from the configuration of publishers and subscribers since the configuration for pub-sub is handled by a user, and addition of another node potentially means modification of multiple files in the code-base. Therefore, it is possible to assume that the **flexibility** of infrastructure is limited. Another issue is due to ROS's pub-sub infrastructure is designed to operate within a single computer [33, 36]. Hence, the current infrastructure is also limited in terms of **horizontal scaling**. The last issue is due to monitoring. **Monitoring** is crucial for company to see if the processing nodes are operational. There is an existing monitoring infrastructure that listens to the published messages and checks the system health based on publications. Similarly the limitation with this approach is also due to **flexibility** since for new components to be monitored, the health check infrastructure requires a new entry within the configuration file. Due to the aforementioned restrictions, the company decided to seek for a better solution for pipelining. The specifications require a WMS that can run natively at initial stages, has an option to integrate with Kubernetes in a future use case and ideally use Python Programming Language.

Chapter 4

Methodology

This chapter will discuss some of the existing workflow management systems, how to select in between the existing tools, as well as a summary that brings an overview for the discussions that were made. The chapter will also detail the architecture and working principles of the selected WMS in detail.

The existing WMSs are vast in numbers, generally implemented by different tech-giants or software foundations and each having different working principles to satisfy the common principle of workflow management [10,30]. To list a few, [10] talks about the following options: Apache Airflow, Argo Workflows, LinkedIn Azkaban, Netflix Conductor, Spotify Luigi, Make, Netflix Metaflow, NSA Nifi and Apache Oozie. A quick online search could also yield other tools such as Kubeflow, PrefectHQ Prefect and Pachyderm. To limit the options, Apache Airflow and five of these options were chosen in random. The WMS selection is like in the following: **Apache Airflow, Spotify Luigi, Kubeflow, Argo Workflows, PrefectHQ Prefect and Pachyderm**.

The reasoning behind to isolate Airflow is due to the immense amount of literature surrounding the WMS directly, as can be observed on Section 3.2. Even for the literature that does not focus on Apache Airflow explicitly, such as [9, 10, 24, 28][1], the tool is frequently mentioned.

As discussed on Section 3.2 selection of an appropriate WMS is a challenge by it's own since the selection problem could be considered as an optimisation problem. It was also discussed that a criteria will play a crucial role for the selection of the WMSs, and the criteria proposed by Kortelainen will be used to make such a selection. Kortelainen divides their strategy into two categories, and each category contains a multitude of sections that can be used for WMS selection [30]. The two main categories are like in the following: *criteria based on the project view* and *criteria based on the technical view* [30]. A brief overview of the criteria provided by Kortelainen can be observed on Appendix A.

[1]The citations refer to the books themselves.

4.1 Project View of WMS Selection Criteria

Project view provides criteria for determining which WMS will comply with the requirements of a given project and the system that the project will run on [30]. It is also possible to perform a quick elimination on project view criteria by determining the priority of categories with respect to the needs of the project. [30]. With respect to the information provided, it is possible to list the criteria with respect to the order of importance for this thesis, as well as the requirements of Section 3.3: installation, community, source code and release maturity, licensing, interface availability and documentation.

4.1.1 Installation

Installation determines where the WMSs can be installed, as well as dependency on other software or platforms to operate, and existence of an official Docker image [30]. The findings for the WMS selection used on this thesis can be summed up on the Table 4.1

WMS Name	Platforms	Dependencies	Docker Image
Apache Airflow	POSIX-compliant Operating Systems, Kubernetes	PostgreSQL or MySQL, SQLite (for testing only)	Exists
Spotify Luigi	Python interpreter	Variety of Python packages	Exists
Kubeflow	Kubernetes	-	Exists
Argo Workflows	Kubernetes	-	Exists
Prefect	Python interpreter	Variety of Python packages	Exists
Pachyderm	Kubernetes	-	Exists

Table 4.1: Installation summary for the selected WMSs. Details obtained from different sections of [37–42] as well as [30] for Airflow and Argo Workflows.

For Luigi and Prefect, term Python interpreter refers to any environment that can run a Python interpreter, since WMSs in question are Python packages by design [38,41]. Similarly, the dependencies for Luigi and Prefect are obtained through the *setup.py* and *requirements.txt* files on GitHub[2] [3].

Another aspect that should be discussed is the term *POSIX-Compliant Operating System* that was used in the Airflow's documentation. POSIX, in brief words, is a standard for defining a common interface in between applications and the operating system, as well

[2] `https://github.com/spotify/luigi/blob/master/setup.py`
[3] `https://github.com/PrefectHQ/prefect/blob/main/requirements.txt`

as some standards in operating system shells [43, 44]. According to the perquisites [45], the term refers to modern GNU/Linux and Linux Distros, Mac OS as well as Windows Subsystem Linux 2.

All of the options were observed to contain Official Docker images. Docker is a popular platform for making applications into isolated portable runtime environments, called *containers*, which eases the deployment and configuration of the programs on host systems [46]. As per Section 3.3, existence of an officially supported Docker image is crucial for the future development plans.

Prefect and Luigi's dependence on python packages do not pose a significant risk since the installation is handled automatically and the projects are still maintained to this day. Airflow's dependency on MySQL or PostgreSQL introduces an additional challenge of configuration of the respective databases, however due to the extensive documentation [37], it can be assumed that the issue is minuscule.

Three of the WMSs rely on Kubernetes to function. With respect to Section 3.3, as well as the requirements of this thesis, preparation of Kubernetes is out of scope, therefore the options Kubeflow, Argo and Pachyderm can be eliminated. However it is important to note that [39, 42] consider machine learning and data science applications as a use case, hence given enough resources Kubeflow and Pachyderm will be options to look at.

4.1.2 Community

Community refers to the popularity of WMSs within online mediums [30]. Community is a essential aspect of the selection considering that Airflow, Luigi and Prefect are open-source projects[4] [5] [6]. Open source advocate Eric Steven Raymond argues that a crucial part that makes an open source project successful is a large community that is working on the same software [47], which eventually became an inevitable fact nowadays.

Kortelainen considers four aspects to evaluate community. First is availability of the community pages for the WMSs. Each of the tools has been observed to have a community page, Luigi being the most limited out of three with reserving only the *issues* section on GitHub as a form of community support [48–50]. Second is the quantity of GitHub Stars, contributors and releases. Third is the available StackOverflow questions and the fourth is the total number of Google Hits. Google hits has been replaced with Google Trends Interest Score. To calculate the interest score, the search volume has been normalised into a score in between 0 and 100, 100 representing the peak interest for the search term [51]. The finding for GitHub star history and interest score can be visualised

[4] `https://github.com/apache/airflow`
[5] `https://github.com/spotify/luigi`
[6] `https://github.com/PrefectHQ/prefect`

on Figure 4.1.

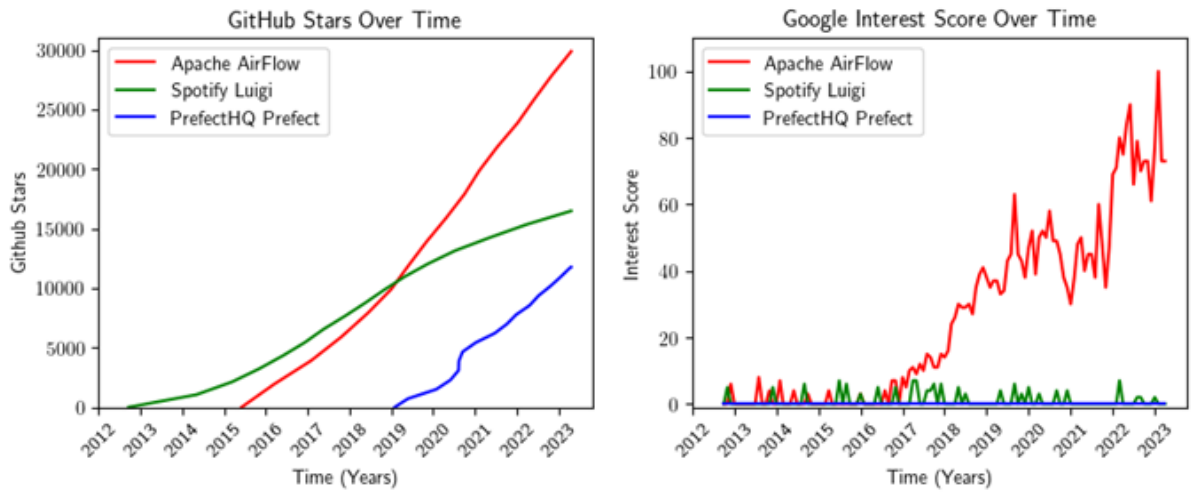


Figure 4.1: GitHub Stars and Google Interest Scores over time. Data obtained from [52, 53]

Similarly the findings for GitHub contributors and releases, as well as StackOverflow questions can be observed on Table 4.2. Values for GitHub were obtained directly from the respective Github pages. The values for the StackOverflow questions are obtained through the StackExchange Data Explorer's query interface [54], running the following SQL query, replacing *WMS* with *airflow, luigi* and *prefect* respectively:

```
SELECT COUNT(*) FROM Posts WHERE Tags LIKE '%WMS%' OR Title LIKE '%WMS%';
```

Where *Tags* represent the tags on the post and *Title* is the title of the post. The reason for this choice is, in theory, to select the posts that are directly on the WMS.

WMS Name	StackOverflow Questions	GitHub Contributors	GitHub Releases
Airflow	9821	2,467	64
Luigi	357	501	56
Prefect	181	149	184

Table 4.2: WMSs and the count of StackOverflow questions, GitHub Contributors and GitHub Releases.

As per Figure 4.1, Airflow clearly surpasses the other WMSs, including Luigi which was released a couple of years earlier than Airflow. Similarly, the interest score indicates that the interest on Airflow is steadily increasing where the interest dwarfing the interest on the other WMSs in comparison.

Similarly, Airflow has significantly more questions asked directly about the tool itself on StackOverflow, as well as a vast quantity of contributors on GitHub. Prefect has

the most releases on GitHub amongst the three options. The reliability of the release as a metric is questionable since release policies vary from project to project. However, frequency of releases could provide an insight on the maturity of the project.

4.1.3 Source Code and Maturity

Source code and Release maturity deals with the availability of the WMS source code, the programming languages that were used in the creation of the WMS, if the WMS's are ready to be used for a real-life use case, and finally the age of the WMS [30]. The subject also goes hand-in-hand with the discussion made on Section 4.1.2 due to the nature of the WMSs, being open source. Therefore, within the context of this project Sections 4.1.2 and 4.1.3 have equal priorities.

The details surrounding the WMSs can be summed up on Table 4.3:

WMS Name	Programming Language	Maturity	First Release
Airflow	Python	Production	August 2015
Luigi	Python	Production	May 2014
Prefect	Python	Production	March 2019

Table 4.3: WMS Programming Language, Maturity and First Release. The programming language and first release dates has been obtained from GitHub repositories (see Section 4.1.2) of the WMSs. Production label indicates that WMS under question has seen use under tremendous workloads [30].

All of the projects are witten in Python programming language, which is a criterion depicted in Section 3.3. Therefore programming language cannot be used as a metric for elimination.

Airflow began it's life as a project within Airbnb and later on joined to the Apache Software Foundation[7] [37]. As discussed at the start of Section 3.2, the literature and work done using the WMS is also quite immense. Henceforth it is possible to assume that Airflow is a production level project. Similarly, Luigi is a tool that is created by Spotify for internal use [38], and with respect to Section 4.1.2 the tool appears to be relatively popular due to GitHub stars, as well as having a relatively high amount of StackOverflow Questions and GitHub contributors. Henceforth it is also possible to assume that Luigi is production level. Lastly, although Prefect lacks the popularity, their website hints at tech-giant business partners[8], which makes it safe to assume that Prefect is also Production level.

[7] `https://airflow.apache.org/docs/apache-airflow/stable/project.html`
[8] `https://www.prefect.io/`

Last aspect to discuss is the first release dates. Luigi and Airflow are the oldest options, project ages being 9 and 8 years respectively. Prefect in comparison is relatively young with only 4 years. With respect to Raymond's argument in Section 4.1.2, it can be assumed that a popular open source project would be more refined compared to other options, due to more individuals having the opportunity to contribute within a greater timespan.

Combining the details discussed on Sections 4.1.2 and 4.1.3, it is possible to eliminate Luigi and Prefect from further evaluation. This is due to Airflow having a significantly bigger community, as well as carrying the adequate maturity characteristics.

4.1.4 Licensing, Interface Availability and Documentation

The remainder of the criteria for project view will be discussed under a single section, due to the discussions being relatively short. As a brief summary, it is possible to say that Airflow does not pose any limitations in regards to the criteria that will be discussed.

Licensing is concerned with the software publication license on the WMS, as well as the potential limitations brought by the license [30]. Airflow is subject to Apache License Version 2.0[9]. The license does not pose any limitations to the use case depicted on Section 3.3.

Interface availability considers the availability of a command line interface (CLI), a graphical user interface (GUI), and a web interface (REST API) which allows user interaction with the WMS [30]. Airflow does contain facilities for the specified interfaces[10] [11] [12]. It is also important to state that during the implementation and evaluation, CLI and GUI were routinely used.

Documentation is about the availability of documentation for workflow development, deployment of the WMS to a production environment (environment where a software is actually put in use [55]), documentation for the architecture, as well as documentation for developers who want to contribute to the platform's development [30]. Airflow documentation [37] provides a detailed insight on the criteria under question, hence documentation aspect has also been found sufficient. Documentation for workflow development is discussed on Section 4.1.1, and architecture is discussed on Section 4.1.2. Documentation for deployment and platform development are available at[13] [14].

[9] `https://airflow.apache.org/docs/apache-airflow/stable/license.html`

[10] `https://airflow.apache.org/docs/apache-airflow/stable/cli-and-env-variables-ref.html`

[11] `https://airflow.apache.org/docs/apache-airflow/stable/ui.html`

[12] `https://airflow.apache.org/docs/apache-airflow/stable/stable-rest-api-ref.html`

[13] `https://airflow.apache.org/docs/apache-airflow/stable/administration-and-deployment/production-deployment.html`

[14] `https://github.com/apache/airflow/blob/main/CONTRIBUTING.rst`

4.2 Technical View of WMS Selection Criteria

Technical view discusses criteria for making an in-depth analysis of the chosen WMS to observe details regarding the design and working principles [30]. In addition to the discussions, some aspect such as definition of the workflows on code-level and details of the software architecture will also be explained in depth to give a better insight on Airflow.

4.2.1 Development of the Workflow

This section will consider three aspects regarding the coding to define the workflow: how are workflows defined, how the tasks are implemented and if there are any extensions for integrated development environments (IDE) [30]. The worflows in Airflow are defined as DAGs, where each vertex is a *Task* [56]. A task is the basic unit of computation and in total there are three types [57]:

- **Operators:** predefined template objects that can be used within the different parts of the DAG.
- **Sensors:** A sub-type of the Operator which wait for external events to happen.
- **TaskFlow:** A function decorator that allows the packaging of python function.

Throughout the project, only the operator type tasks were used through the utilisation of PythonOperator, which allows the execution of Python functions [58]. An example for two DAGs can be observed on Appendix B. First DAG is a simple implementation with two Python Operators executing the same function and the second is a scaled-up version of the initial implementation with the addition of a third operator.

In terms of the IDE extensions, two most popular options for Python development were considered: VSCode and PyCharm. VSCode does contain an extension for Airflow[15]. PyCharm does not support any extensions for Airflow.

4.2.2 Architecture

Kortelainen [30] considers the four criteria for the evaluation of the WMSs. The first is the *high availability*. Availability is a software term used to describe the possibility that a system or a program will be operational within a given time period [17]. The second is the scalability of architecture. Third is *state persistence*, which is another software

[15] `https://marketplace.visualstudio.com/items?itemName=NecatiARSLAN.airflow-vscode-extension`

term used to describe a program's or a system's ability to return to a certain state (of task execution) [17]. Last one is the support for asynchronous messaging. Before making any further discussions regarding the criteria, it is also important to understand the operational principles of Airflow architecture. A brief summary can be observed on the Figure 4.2:

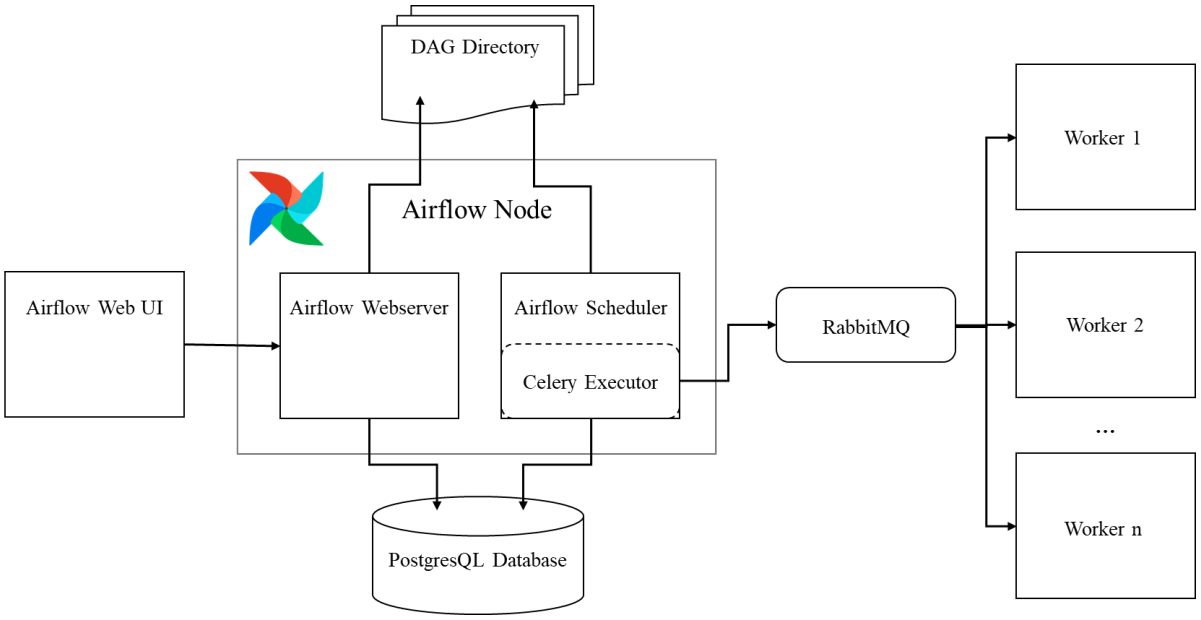


Figure 4.2: Architecture diagram of Airflow, depicting the implementation described in Section 5.4. The diagram is based on the diagram in [59].

There are five main components of the Airflow architecture:

- **Scheduler:** Triggers the scheduled workflows as well as submitts tasks for the executor to run [59].
- **Executor:** Handles the running/execution of the tasks provided by the scheduler. The setup created utilises *Celery Executor* and *RabbitMQ* for the execution of the tasks. Celery creates a client (Airflow), workers for task execution, and handles the communication over message queues (RabbitMQ) [60]. Utilisation of Celery provides the ability of parallel and concurrent task execution[16] and alternative choices for Celery are available[17].
- **Web Server:** Provides a GUI in the form of a web page. The UI allows triggering, inspection and debugging of the DAGs [59].
- **DAG Folder:** Contains Python scripts defining the workflows/pipelines [59].

[16] `https://airflow.apache.org/docs/apache-airflow/stable/core-concepts/executor/celery.html`

[17] `https://airflow.apache.org/docs/apache-airflow/stable/core-concepts/executor/index.html`

- **Metadata Database:** Stores states of scheduler, executor and web server [59]. Metadata database can also be used for the interprocess communication through the use of XComs[18], which were also used within the implementation for data accumulation (see Section 5.2). PostgresQL was used for the implementation due to the handling of concurrent access[19].

Kortelainen argues that the architecture is limited in terms of availability due to the infrastructure relying on a single scheduler [30]. However during the evaluation, it has been observed that if any of the aforementioned components fail, only exception being the web-server, the execution of DAGs will stop. Hence, the argument for availability is up for discussion since the same argument can also potentially apply for any piece of software relying on a multitude of components, which involves the other WMSs.

Testing can be done for availability, which is out of the scope for this thesis. Furthermore, due to the popularity of the framework as per Section 4.1.2 it is safe to assume that the software does not face any critical issues surrounding availability.

On Section 4.2.1 it has been argued and demonstrated that Airflow DAGs are created with modular components that can be added or removed from DAGs depending on demand (flexibility). On Section 4.1.1 it has also been discussed that Airflow can operate on Kubernetes. Furthermore, Harenslak and de Ruiter [10] also claim that Airflow can operate on horizontally scalable systems. Therefore it is possible to assume that Airflow is scalable both in terms of software design and within it's implementations on distributed computational platforms.

The metadata database's primary function is to store states for different components, hence Airflow has state persistence.

The details for asynchronous messaging have not been discussed in detail by Kortelainen, but it can be assumed that the concept refers to data transfer for interprocess communication and logging purposes. The existence of XComs also allows interprocess communication. Airflow also contains callbacks[20] for logging purposes which are asynchronous by design. Therefore, it is possible to assume that Airflow does provide asynchronous messaging facilities.

4.2.3 Workflow Features and Control Mechanisms

Kortelainen uses five different metrics to determine workflow features and control mechanisms of the WMS: existence of dynamically advancing graphs which are graphs capable

[18] `https://airflow.apache.org/docs/apache-airflow/stable/core-concepts/xcoms.html`
[19] `https://www.postgresql.org/docs/current/mvcc.html`
[20] https://airflow.apache.org/docs/apache-airflow/stable/administration-and-deployment/logging-monitoring/callbacks.html

of adding tasks during executions, availability of a scheduling mechanism, a form of access management, existence global semaphores, and limiting of the execution rate [30]. The criteria surrounding global semaphores were not clear. A semaphore in a general sense could be described as variable for the synchronisation of program running tasks concurrently [17]. There is a problem due to the definition since Airflow (when configured with Celery) executes tasks both parallel and concurrent (see Appendix C). Therefore, after a brief exchange, Kortelainen described global semaphores as an internal limiting within the WMS that determines the number of parallel executions of Tasks.

As per Section 4.2.2 the architecture of Airflow already facilitates a scheduling mechanism in the form of a scheduler. The scheduler also goes hand in hand with the rate limiting since an argument of a cron expression[21] can be passed to the DAG[22], determining the time intervals for DAG executions. An example can be observed in the code snippet bellow, where the DAG specified in Appendix B is introduced with an scheduling interval, which executes the DAG at 4:05AM:

```
1 dag = DAG(
2       dag_id='0_example_dag',
3       default_args=default_args,
4       schedule_interval="5 4 * * *",
5       max_active_runs=1
6 )
```

The Web UI facilitates a form of access control in terms of creation of individual users with different permissions[23].

There are discussions surrounding dynamically advancing graphs. Kortelainen argues that the dynamically advancing graphs could be implemented in an indirect way [30]. However during investigations it was discovered that dynamically advancing graphs[24] were introduced on Airflow version 2.3.0[25] which was available after Kortelainen's work was published. Regarding the global semaphores, Kortelainen argued that no such mechanism existed on Airflow [30], it has been observed that a limitation of parallel tasks is possible through the executor configuration. An example for Celery executor can be observed on Appendix C. Therefore it is possible to conclude that the most recent version of Airflow available supports dynamically advancing graphs and global semaphores, however global semaphores may be limited since the criterion relies heavily on the executor choice.

[21] https://crontab.guru/

[22] https://airflow.apache.org/docs/apache-airflow/1.10.1/scheduler.html

[23] https://airflow.apache.org/docs/apache-airflow/stable/administration-and-deployment/security/access-control.html

[24] https://airflow.apache.org/docs/apache-airflow/2.3.0/concepts/dynamic-task-mapping.html

[25] https://airflow.apache.org/blog/airflow-2.3.0/

4.2.4 Application Delivery, Code Reuse, Available Integrations

The criteria will be discussed within a single chapter since the discussions surrounding are brief and some are not a direct concern for the specifications depicted on Section 3.3.

Application delivery is concerned with the automated deployment of Airflow to Platforms such as Kubernetes or the availability of Continuous Integration/Continuous Delivery (CI/CD) tools [30]. CI/CD is a term referring to automation of testing and deployment of code [61]. Deployment automation is available for Kubernetes in the form of a Helm chart[26]. Helm is a package manager for Kubernetes for making software built on Kubernetes available [62]. Airflow itself does not contain an out-of the box tool for CI/CD automation, however considering the definition of the term, it is possible to assume that a solution is available.

Code reuse is concerned with the availability of workflow examples and code samples in general [30]. Airflow has an extensive online documentation [37] that covers different aspects of the implementation. As per Section 3.2 and Section 4.1.2 the existing literature around the WMS and the community is vast, providing additional samples of workflow and code.

Available integrations is concerned with the possibility to integrate the tool with cloud environments and different software platforms [30]. Airflow documentation also contains sections for cloud integrations[27] and a selection of provider packages for integrations with other platforms[28].

4.3 Chapter Summary

Out of the six WMS options, Airflow has been selected as the WMS that will be used for the modelling of the pipeline described in Section 3.3. Selection has been done using the criteria proposed by Kortelainen [30] (see Appendix A). First a quick elimination has been performed using the project view criteria, and Airflow became the WMS of choice due to it's ability to operate on a multitude of platforms, having a large community and carrying the characteristics of a mature software. The rest of the project aspects were also considered to observe if Airflow had limitations on licensing, documentation and interface availability, and none were discovered. Then a thorough technical view for the WMS has been conducted to investigate various details regarding code development, WMS architecture, featues of the workflow and available control mechanisms, application delivery methods, code examples, and available platform and cloud integration options. Some

[26] `https://airflow.apache.org/docs/helm-chart/stable/index.html`
[27] `https://airflow.apache.org/docs/apache-airflow/1.10.10/integration.html`
[28] `https://airflow.apache.org/docs/#providers-packages-docs-apache-airflow-providers-index-html`

minor concerns were discovered regarding the availability of the tool and a lack of out-of-box application delivery environments has been observed. However it was concluded that the concern does not pose a risk considering the requirements made on Section 3.3. An overview of the findings can be observed on Table 4.4:

Category		Remarks
Project view	Licensing	Has a open license.
	Installation	Can run in a multitude of platforms, which includes a host operating system, Docker and Kubernetes.
	Source Code and Maturity	Project has been actively developed for 8 years, Python is the main language.
	Community	Has a really large community.
	Interface Availability	Supports a CLI, a GUI and a REST API.
	Documentation	Has an extensive documentation.
Technical view	Development	Workflows are defined as DAGs and implementations are scalable has extensions for some IDEs.
	Architecture	Minor concerns for availability, has scalable architecture, has state persistence and supports interprocess communication.
	Workflows	Contains adequate features and control mechanisms for scheduling limiting, access control and dynamic increase of tasks.
	Application Delivery	No out-of-the-box options for automated application deployment, or CI/CD environments.
	Code Reuse	Plenty of examples for code development.
	Available Integrations	Supports cloud and platform integrations.

Table 4.4: An overview of Kortelainen's WMS selection criteria (see Appendix A) for Airflow.

Chapter 5

Implementation

This chapter will discuss a model implementation made, using Airflow, for the data pipeline described on Section 3.3, how the data is gathered from the implementation, the assumptions that were made while designing the model, and finally the details of the setup that was created to test the implementation.

An important aspect to note is that the model and the implementation is a mock scenario, rather than an actual implementation processing data. The choice is intentionally made to provide a proof-of-concept (PoC). Although in Chapter 4 a thorough research on Airflow has concluded that the WMS is a capable tool to undertake a variety of pipeline orchestration challenges, it is not guaranteed that the software is the best match for the use case depicted on Section 3.3. Hence PoCs are essential to demonstrate if a hypothetical application of the WMS has a merit [63].

5.1 Mock Scenarios

In total there has been two scenarios that has been created with respect to the specifications of the use case described on Section 3.3. Scenarios have three *layers* (different processing stages of the pipeline), consisting of an extract-transform layer, a feature processing layer and finally a classification layer. Each layer is consisted of a multitude of dependencies, as per it's real-life counterpart. The only difference between the scenarios are the number of tasks, hence the tasks are named *Small DAG* and *Large DAG* respectively. Small DAG acts as a base case scenario, replicating some of the processes running on the WasteAnt back-end. Large DAG is an extension on top of the Small DAG, to replicate the scenario of another classification pipeline being added in parallel to the main pipeline. A brief overview of the scenarios can be observed on Figure 5.1 and the code for large DAG can be observed on appendix D:

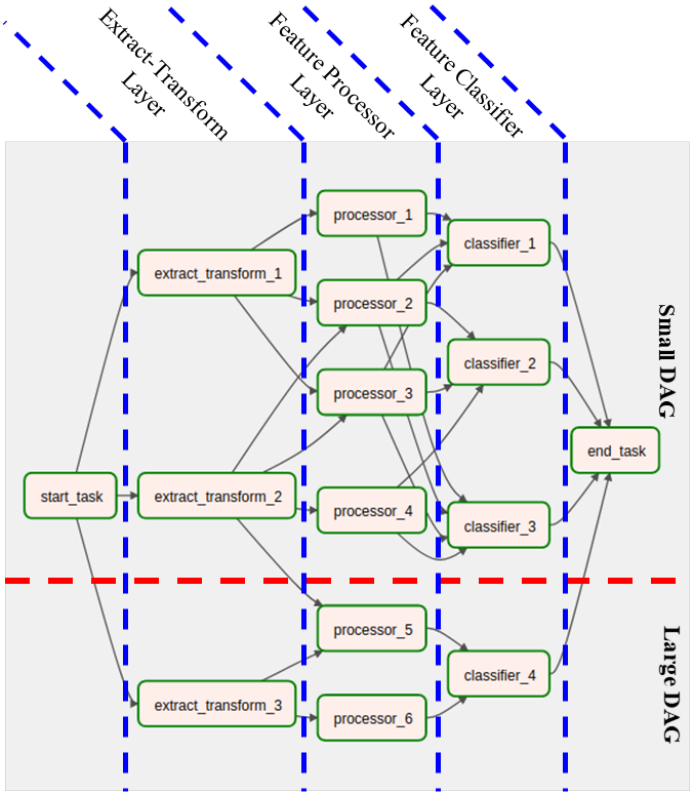


Figure 5.1: Illustration of the DAGs, replicating the Use Case depicted on Section 3.3. Base diagram is obtained through the Web UI of Airflow. Red line separates the contents of Small and Large DAGs, Large DAG contains all the items. Blue line separates the processing layers.

As can be observed, Figure 5.1 demonstrates processing layers, tasks within the layers an the dependencies of the tasks/upstreams of the tasks. Start and End tasks are passive tasks for measuring the processing time, as well as logging the data obtained from the DAG execution. Small DAG is consisted of two extract-transform (data pre-processing) tasks, four processor tasks and three classifier tasks. Large DAG is a scaled-up version of the Small DAG, containing an extra extract-transform task, two processor tasks and a classifier task.

Another important aspect to note is the way tasks simulate the data processing happening on the use case. Each task during their execution performs two functions. First function is the generation and logging a random number in between a pre-defined interval. The number represents the input data amount in megabytes processed by the task. The second function is the generation of another random number to put the task in sleep. Sleeping represent the amount of time required for data processing. The concept of functionality can be summed up on Figure 5.2:

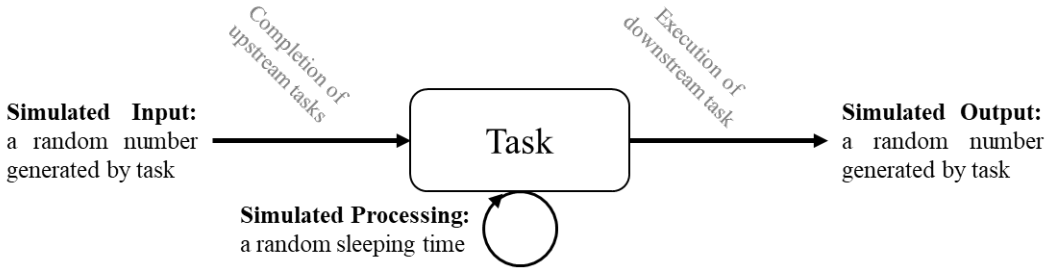


Figure 5.2: Simulation of the data processing happening inside DAG tasks.

There are three different types of simulation parameters, each corresponding to their respective layers of execution. The random intervals are based on the observations made on the use case and can be listed like in the following:

- **Extract-Transform Tasks:** Processes data that are roughly in the range of **800 to 1200 megabytes**. Typically takes **10 to 15 minutes to process**.
- **Feature Processor Tasks:** Processes data that are roughly in the range of **600 to 750 megabytes**. Typically takes **25 to 30 minutes to process**.
- **Classifier Tasks:** Processes data that are roughly in the range of **400 to 600 megabytes**. Typically takes **40 to 45 minutes to process**.

There were two types of DAG executions done in total. First a simple sequential execution of tasks with respect to their priority, where all upstream tasks are executed before their downstream. Simple execution has been done to create a demonstration case for the pipelines potential worse case runtime. Second one is an execution of tasks through a production-ready Airflow implementation, where concurrency and parallelisation is achieved through Celery Executor.

5.2 Collection of the Data

Two types of data were collected from the implementation for evaluation purposes. First one is the total amount of the data that has been processed by the tasks, which is referred as *throughput* on the code. Second one is the time interval that the execution was completed, which is referred as *record time* on the code.

Data has been recorded discretely in 15 minute intervals. Discrete data collection has been done to demonstrate the distribution of the total data size that has been processed by the tasks in given intervals (see Figures 6.1 and 6.2). A pseudo-code can be observed on the following snippet, demonstrating how each task recorded the data amount processed within a given time interval to the XComs (see Section 4.2.2 and appendix D):

```
def mimic():
 generate data amount and sleep interval
 sleep for sleep interval
 calculate elapsed time since sleep
 fetch the DAG execution start time
 for interval in time intervals
  if elapsed time - DAG start <= interval
   xcom_push(key: task name + "_data", value: data amount)
   xcom_push(key: task name + "_record_time", value: interval)
```

As per Section 5.1, there are start and end tasks. Start task only records the start of the DAG run. End task checks all the entries on XComs to provide an output of the recorded data in the form of a log file. A pseudo-code for the end task can be observed on the following snippet:

```
def end_task():
 fetch recorded data from XComs
 total_data = 0
 for task in tasks:
  total_throughput += task["throughput"]
  log(Total throughput: total_data,
   record time: task['record_time'],
   task: task['task'],
   throughput: task['data'])
 log(Total throughput: total_data)
```

It is also important to note that the total data size is logged for each DAG execution to demonstrate the accumulation of data size over time.

5.3 Assumptions and Limitations

In total, there are three assumptions made during the modelling, which in some cases introduces a series of limitations for evaluation. The first assumption is made on the delivery of data, in which it is assumed that the implementation is already delivering data in a desired schema. As per Section 4.2 Airflow handles orchestration in the form of defining a DAG for Tasks running a Python function. Hence the model represents a hypothetical implementation where the code base of the use case is put into separate functions to be orchestrated, in which the data schema already matches the expectations.

The second assumption is on the runtime, in which the actual runtimes were in seconds

rather than hours and minutes. Since the tasks are not processing real data, transliterating time data from seconds to minutes does not bring any limitations for the experimentation. Therefore the actual runtimes were kept short in order to leave space for multiple trials.

The third assumption is for the serial execution. During the trials, Airflow's sequential executor has been utilised to simulate the worst case runtime. Due to the nature of the task implementations, the choice for orchestration is not significant as long as a single task is processed at a given time. Another aspect to note is that coming up with orchestration solutions are also a programming challenge which requires investment of a significant amount of time. Hence the sequential execution is also simulated through Airflow.

The last assumption is regarding the behaviour of the data sizes. The model uses Python's pseudo number generator, which generates data in a uniform distribution. A histogram generated with Python's random number generator can be observed on Figure 5.3.

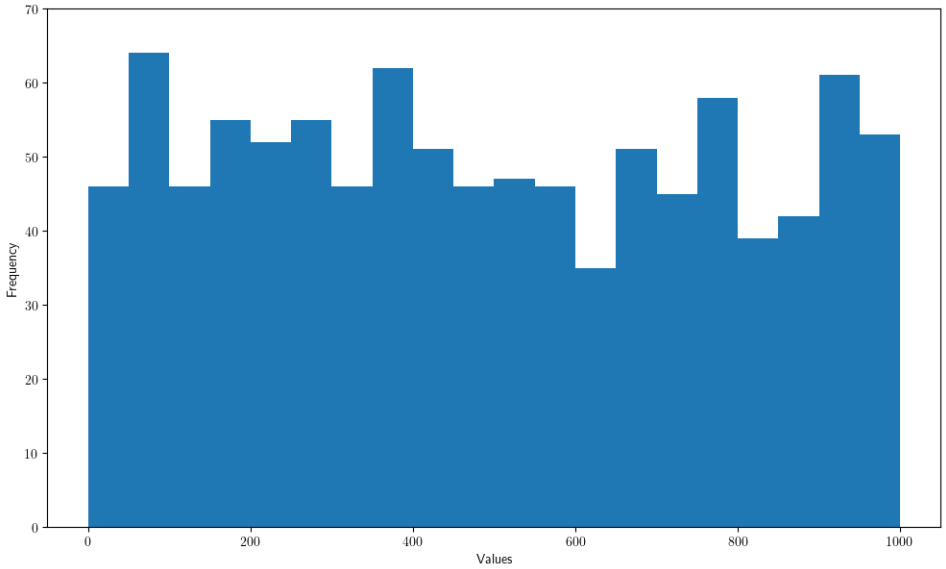


Figure 5.3: Histogram generated by Python's random number generator. Histogram displays numbers generated randomly in between 0 and 1000, with the bin size of 50.

As can be observed, the distribution of the values tend to be within a uniform range, hence bringing data no noticeable statistical characteristics. The choice was intentionally made in order to simulate a case where data sizes within the range are not predictable, or in other words to mimic a worse-case scenario. Although the approach replicates a worse case situation, a potential limitation is introduced with the accuracy of data in real-life.

5.4 Experimental Setup

The testing of the model has been done on a virtual machine located on a host computer. The details surrounding the Airflow implementation, virtualisation software and host system hardware can be observed on Tables 5.1, 5.2 and 5.3.

Regarding the virtual machine and hardware specifications, only details were only provided for specifications that might directly impact the performance of the runtime environment. Additionally, the laptop PC was plugged in to a power outlet during the execution of the trials to prevent performance loss due to power saving.

Component	Version
Airflow	2.5.3
Celery Executor	5.2.7 (Dawn Chorus)
RabbitMQ Docker Image	3.11.13
PostgresQL	14.7 (Ubuntu 14.7-0ubuntu0.22.04.1)
Python	3.10.6

Table 5.1: Details of the Airflow implementation, depicted on Figure 4.2.

Specification	Details
Virtualisation Software	VMware® Workstation 17 Player
Virtualisation Software Version	17.0.0 build-20800274
Virtualised OS	Ubuntu 22.04.2 LTS
Assigned CPU Threads	6
Assigned RAM	16GB
Assigned Storage Space	100GB

Table 5.2: Details of the virtualisation environment hosting the runtime environment of Airflow. Word *Threads* refer to virtual threads provided by Intel® Hyper-Threading Technology[1].

[1] `https://www.intel.com/content/www/us/en/gaming/resources/hyper-threading.html`

Specification	Details
Computer Model	Dell XPS-15 9570 Laptop PC
OS	Microsoft Windows 10 Home 22H2 Build 19045.2846
CPU	Intel® Core™ i7-8750H Processor, 2.20GHz up to 4.10GHz, 6 Cores 12 Threads
RAM	2 x Corsair Vengeance Performance 16 GB 2666 Mhz DDR4 CL18, 32 GB Total
Storage	Samsung PM981 Polaris 1TB M.2 NGFF PCIe Gen3 x4, NVME SSD

Table 5.3: Details of the host PC. Two clockspeeds listed on the CPU details refer to Intel® Turbo Boost Technology[2].

[2] `https://www.intel.com/content/www/us/en/architecture-and-technology/turbo-boost/turbo-boost-technology.html`

Chapter 6

Evaluation and Conclusion

This chapter will discuss the evaluation made for the implementation described in Chapter 5. The evaluation will provide an insight on if Airflow is an efficient WMS for the use case, using the following arguments as a basis: processing time, data throughput, scalability and monitoring mechanisms. As per Section 5.1 two DAGs were taken in consideration and two execution methods were done. For each execution ten trials were made. Therefore, for each DAG twenty trials were made overall and since there are two DAGs forty trials were made in total. The processed data amounts versus time is illustrated on Figures 6.1 and 6.2 and the raw results can be observed on Appendix E.

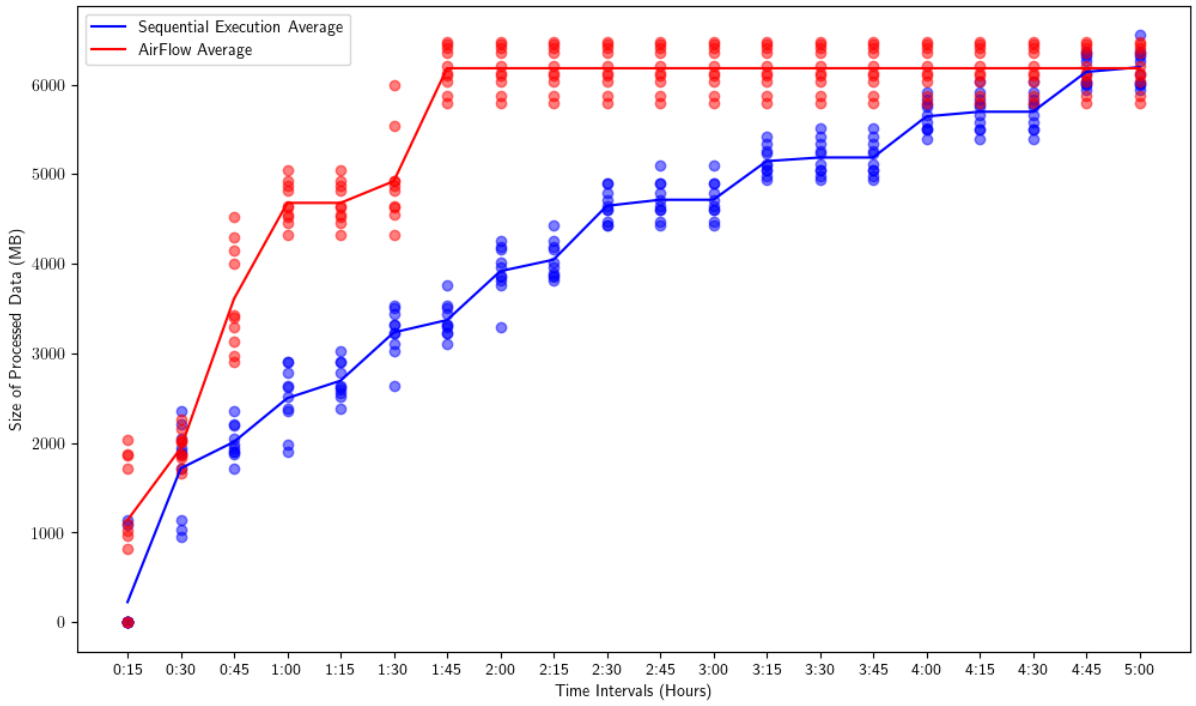


Figure 6.1: Size of processed data versus processing time for small DAG. Dots represents the recorded data amounts at a given interval; colour red is for Airflow and blue is for sequential execution.

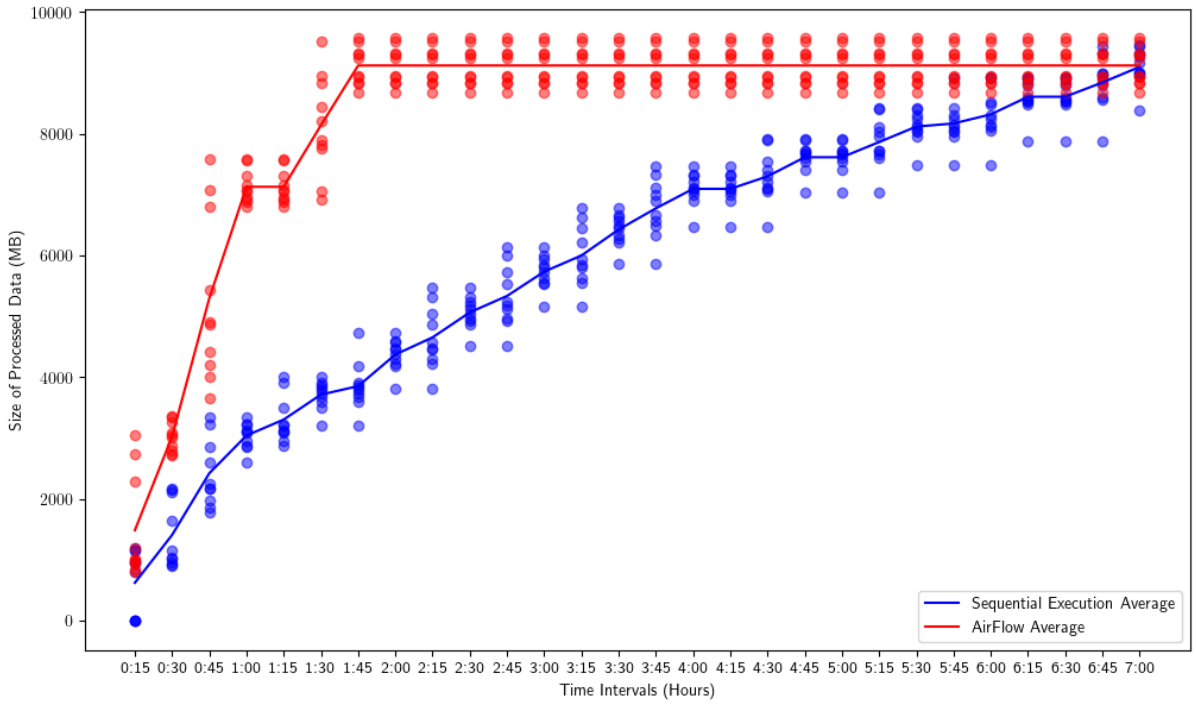


Figure 6.2: Size of processed data versus processing time for large DAG. Dots represents the recorded data amounts at a given interval; colour red is for Airflow and blue is for sequential execution.

In Chapter 5, it has been discussed that the data size is randomly generated in predefined intervals based on the observations. In other terms, the randomly generated data sizes mimics a sample from values observed in real life. It was also discussed that the distributions were uniform in nature, to replicate a potential worse case scenario rendering the data unpredictable. In order to perform a parametric statistical analysis, one of the perquisites is to data to follow a normal distribution pattern [64,65]. However the averages of trials per run and DAG can still be taken to calculate the average data throughput, and provide an insight on the data. Average or mean can be mathematically formulated like in the following [65]:

$$For\ samples:\ x_1,\ x_2,\ ...,\ x_n\ (sample\ size\ is\ n)$$

$$\mu(x) = \frac{\sum_{i=i}^{n} x_i}{n}\ where\ \mu\ is\ mean$$

Similarly, with respect to the definition for throughput made on Section 2.5 the average data throughput can be formulated like in the following:

$$T_\mu = \frac{Data_\mu \times 8}{t_\mu}\ where:$$

$T_\mu = average\ throughput\ (in\ Mb/s),\ Data_\mu = average\ size\ of\ processed\ data\ (in\ MB),$

$$t_{\mu} = average\ processing\ time\ (in\ seconds)$$

With respect to the previously made mathematical definitions, the evaluation of the data gathered from the trials can be summed up on Table 6.1:

DAG	Execution	Processed Data (MB)	DAG Runtime (Hours)	Throughput (Mb/s)
Small DAG	Sequential	6192.4	4:45	2.90
Small DAG	Airflow	6181.6	1:35	8.71
Large DAG	Sequential	9098.6	6:48	2.98
Large DAG	Airflow	9121.0	1:34	12.94

Table 6.1: Evaluation of experiments to calculate data throughputs. Processed Data and DAG Runtime are the average values per DAG and Execution. Throughput is calculated from the averages.

6.1 Processing Time

As per Figures 6.1 and 6.2 it is possible to see that Airflow implementation tends to complete tasks in a shorter time period than a hypothetical pipeline executing tasks sequentially. Table 6.1 also display that Airflow on average has a shorter processing time compared to sequential execution.

Another aspect to consider are the *plateau* regions happening at the trend lines of Airflow in Figures 6.1 and 6.2. Plateau occurs in between the 1 hour and 1 hour 15 minute intervals since the start of the DAG executions. The reasoning is due to the runtimes defined on Section 5.1 and the data collection described on Section 5.2. To explain the anomaly in mathematical terms *Big O notation* can be used.

Big O notation in computer science is a formal way to represent runtimes of the algorithms [66]. With respect to the information provided on [66], it is possible to denote the Big O notation of the tasks for the worse-case runtime like in the following:

$$Suppose\ f_t(n)\ is\ a\ fuction\ representing\ runtime\ of\ a\ DAG\ task:$$

$$f_t(n) = O(n)\ where\ n\ is\ the\ worse-case\ runtime\ in\ minutes$$

Using the definition, it is possible to define the Big O for a pipeline section with one extract-transform task and one feature processor like in the following:

$$Suppose\ f_e(n_1) = n_1\ is\ worse\ case\ runtime\ of\ a\ extract\ transform\ task$$

$$Suppose\ f_p(n_2) = n_2\ is\ worse\ case\ runtime\ of\ a\ feature\ processor\ task$$

$$Suppose\ f_s(n) = n\ is\ worse\ case\ runtime\ of\ a\ pipeline\ section$$

$$f_s(n) = f_e t(n_1) + f_f p(n_2)$$

$$n = n_1 + n_2$$

$$= O(n)\ where\ n = n_1 + n_2$$

Using the information provided, the worse case runtime for a pipeline section can be calculated as to be around 45 minutes. Since the actual times are recorded in seconds, due to query operations and time calculations happening within the tasks, a shift can also occur to the next intervals. The shift can be observed with the increase in the data distribution at 45 minute interval of Figures 6.1 and 6.2.

Although a shift in values happen due to a technical error, the existence of plateaus are a natural side effect of combined runtimes of the tasks. The phenomena shows that there is a strict time interval where no additional data can be processed until the executions for a certain processing layer depicted on Figure 5.1 are complete. Considering the DAG implementations in Section 5.1 the phenomena is most likely to be limited to the processors 1 to 4 since the tasks are the upstreams of classifiers.

Another consideration to be made is the fact that plateaus are strictly bound to the processing layers. Considering a scenario where the Airflow implementation is running on a computer with infinite resources, it is safe to assume that as long as the number of layers do not increase the processing time will remain constant. The reason is that Airflow, when configured with different executors, provides parallel and concurrent execution (see Section 4.2.2), meaning all the tasks existing on the same layer can be processed simultaneously.

As a brief summary, it is safe to assume that Airflow is able to compute tasks in a relatively short period of time, due to the facilities provided for parallel execution.

6.2 Data Throughput

As discussed on Section 6.1, there were some inaccuracies caused due to the shifts on processing time measurements. Since processing time is a key components for the calculation of throughput, hence the accuracy of the results are a subject of discussion. Shifts increase the recorded processing time for all tasks. However considering that the same recording mechanism has been utilised for all trials, the effects of the shift could be considered to be neutral in a way that the shift would only reduce the throughput to an extent.

As per Table 6.1 it is possible to observe that Airflow executions outperformed serial execution. For small DAG Airflow had about three times more data throughput and for the large DAG the factor is about four times as much. Therefore it is possible to say that Airflow is capable enough to provide a high throughput rate

6.3 Scalability

Throughout the trials, only some aspects of scalability were observed, in which most were related to the software aspect of scalability. Refering back at the definition made on Section 2.3 and the discussions made on Sections 6.1 and 6.2, Airflow has demonstrated to be capable of handling the increasing workloads. Workload increase has been provided with the large DAG in which four more tasks were introduced. During the evaluations made for the large DAG, it has been observed that Airflow was able to process more data within the same processing interval, yielding in a higher data throughput. Hence it is possible to say that Airflow is a scalable platform overall.

In terms of the scalability of software, it can be observed in appendices B and D that the code is divided up into small modules. As demonstrated in appendix B the modules can be added to the modules can be added to the DAG, without interrupting the workflow. Henceforth it is possible to say that the DAGs created inside Airflow are scalable software components.

The scalability of Airflow on distributed systems has not been considered during the span of the thesis. However, looking at the discussions made on Chapter 4 it can be assumed that Airflow is also horizontally-scalable when being operated on a distributed environment.

6.4 Monitoring

Throughout the evaluation of the experiments, it has been observed that Airflow contained monitoring mechanisms. A visualisation of monitoring can on Figure 6.3. Referring back at the monitoring description made on Section 2.4, the monitoring scheme Airflow employs is capable enough to demonstrate changes of the DAG states. Monitors can also display the stages of the system where failures occur and provide details in the form of an error log. Therefore it is possible to say that Airflow contains a mechanism of monitoring in the form of a graphical interface.

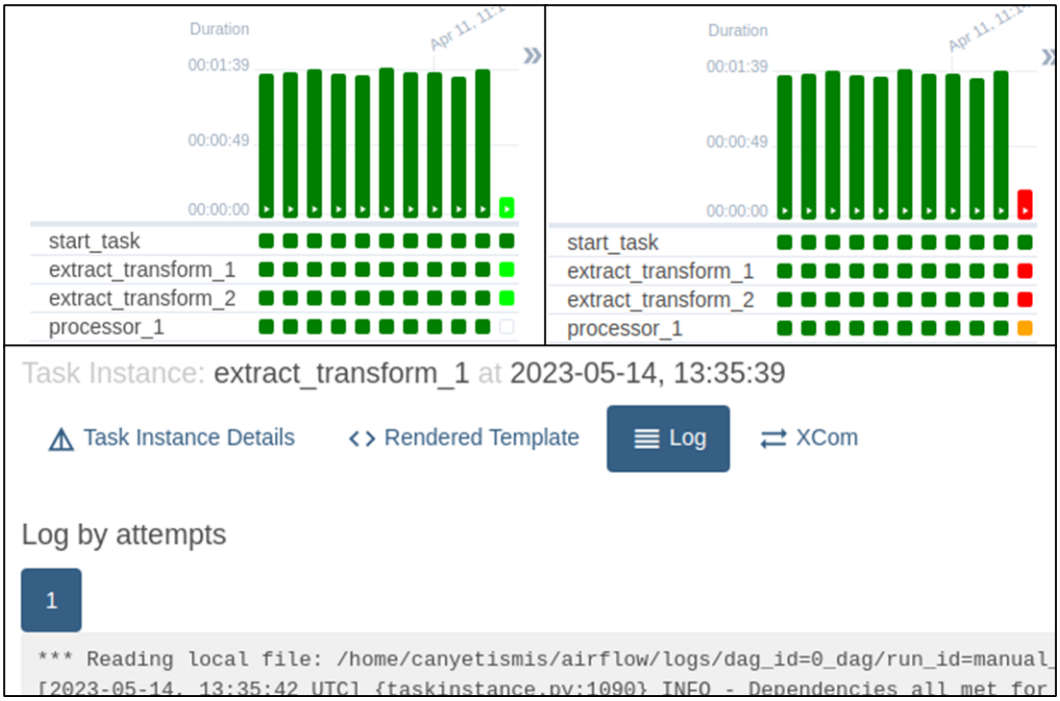


Figure 6.3: Monitoring interface of Airflow. Top left displays completed task executions (dark green) as well as tasks that are under execution (light green). Top right displays a failed task execution (red) and the upstream tasks failed due to a failure in downstream (orange). Bottom displays the error log of the failed task. Colour definitions are obtained from the Airflow interface.

6.5 Conclusion

A thorough analysis has been made on Chapter 4 to determine a suitable workflow management system that could be utilised to solve the problems being faced on the use case described on Section 3.3. In the end Airflow became the WMS of choice for research purposes. Later on two DAG implementations have been made to replicate the pipeline structure of the use case, and a scenario where the infrastructure expands. In total the DAGs were executed with two different strategies: sequential execution to demonstrate the worse-case scenario for the pipeline and a production-ready Airflow implementation to observe the improvements. With each strategy ten trials were made adding up to forty trials in total.

An evaluation has been conducted with respect to the criteria considered in Section 2.5. In Section 6.1 it was discovered that due to a technical error the runtime of the trials were not accurate. Although inaccuracies were present it was discovered that Airflow had the same runtime for both of the DAGs and it was concluded that Airflow had a shorter processing time. In Section 6.2 the average data throughputs per trial and execution were evaluated and it was discovered that Airflow was able to provide a higher data throughput. In Section 6.3 it was argued that due to Airflow being able to provide

higher data throughputs in same processing time when the workload is increased, the WMS was determined to be a scalable tool overall. The coding aspect of scalability has also been considered in which Airflow has demonstrated that the DAG definitions are made with modular components and the components can be added to the DAG without affecting the Airflow runtime. The scalability of airflow on distributed system has not been considered however, the discussions on Chapter 4 also hint at the tool being scalable in distributed platforms. Hence it was concluded that Airflow is scalable both in terms of hardware and software. Lastly in Section 6.4 it was concluded that Airflow contained mechanisms for monitoring that can keep track of the system state, as well as the point of failure.

Considering the definition made on Section 2.5, it is possible to conclude this thesis by saying that Airflow presumably, is an efficient workflow management system for the given application case.

6.6 Future Work

There are a couple of future works that can be done regarding the approach taken in this thesis. The first concerns the Methodology. As described in Section 4.2.1, some options were eliminated due to the lack of computational facilities having a Kuberentes environment. However it was also stated that the documentations of some of the rejected tools explicity mentioning the tools being designed for machine learning pipelines, and the pipeline of the use case is a machine learning pipeline. Hence a potential future work may be done by taking some of the rejected tools in consideration again to investigate different WMS alternatives.

Second set of concerns are the implementation and evaluation. First and foremost, the evaluations were done on mock scenarios replicating the pipelines of the use case. Although the approach is good to demonstrate a proof of concept, no definitive conclusions can be drawn unless the concept is carried to a real life scenario.

Another problem of the mock scenarios was that the fact that it was comparing Airflow to a worse-case scenario, instead of it's real life counterpart. Again, if a real life pipeline were to be implemented using Airflow a more insightful demonstration can be made

In Section 5.2 it was explained that the data has been collected discretely and due to the discrete collection of data some errors were discovered in evaluation. A continuous approach can also be tested in order to see if any improvements will be made.

In Section 5.3 three assumptions proved to be critical while demonstrating Airflow's performance. First one was the assumption that Airflow will not intervene with the desired scheme. Although the assumption is plausible a real life demonstration will be

more insightful.

Second assumption was made while transliterating seconds to minutes and in Section 6.1 it was discovered that there were time shifts happening. Presumably the culprit behind the time shifts is the transliteration so the strategy can be re-evaluated.

Last assumption was made on the simulation of data, in which data sizes were generated randomly. Although the approach is good to demonstrate a scenario were data sizes are completely random within some intervals, the real life counterpart may behave differently. Hence a more thorough investigation can be conducted on data sizes to observe if there are any underlying statistical properties such as a normal distribution.

Chapter 7

Acknowledgements

I would like to thank my supervisor **Dr. Stefan Kettemann** for supervising this thesis, and providing me with a basis for graph theory which was used within this thesis. I would like to thank my second supervisor **Dr. Christian A. Müller**, one of the founders of WasteAnt for providing me with his extensive support throughout the duration of the thesis, whether it being the structure of the thesis or the nature of the research that I was doing. I would like to thank **Dr. Carlos H. Brandt**, working as a researcher in Constructor University, for his support regarding the discussion on scalability, Airflow and my overall thesis structure, as well as recommending *The Cathedral and The Bazaar* at one of our discussions, which eventually ended up becoming a resource in this thesis. I would like to thank **Mr. Panu Kortelainen**, who is working as a consultant in Netlight, for providing an insightful review of workflow management systems, as well as explaining details of his thesis over LinkedIn when I reached out to him.

I would like to thank **Mrs. Lale Evrim Yetişmiş**, working as the CEO/Developer in BE1 Consultancy for providing me a script to transform log files to spreadsheets. I would like to thank **Dr. Ece Uykur**, working as a researcher in Helmholtz-Zentrum Dresden-Rossendorf for providing me with this LaTeX template.

I would like to thank my current institution **Constructor University** for teaching me the disciplines of *Data Engineering* and *Data Science* during my learning journey. I would also like to thank my previous institution, **The University of Nottingham, School of Computer Science** for teaching me the disciplines of *Computer Science* and *Software Engineering*.

Finally I would like to thank my family, my friends and all the people who are dear to me for providing me with emotional support while I was writing this thesis.

Bibliography

[1] TechTarget. DEFINITION zettabyte. Accessed: 2023-05-14. [Online]. Available: https://www.techtarget.com/searchstorage/definition/zettabyte

[2] P. Taylor. Volume of data/information created, captured, copied, and consumed worldwide from 2010 to 2020, with forecasts from 2021 to 2025. Accessed: 2023-05-14. [Online]. Available: https://www.statista.com/statistics/871513/worldwide-data-created/

[3] A. Galkhov, *Probabilistic Data Structures and Algorithms for Big Data Applications*. 12-14 Rond Point Des Champs Élysées, Paris, Ile-de-France, 75008, France: Books on Demand, 2018, p. preface.

[4] P. Sunagar, R. Hanumantharaju, G. Siddesh, A. Kanavalli, and K. Srinivasa, "Chapter 2 - influence of big data in smart tourism," in *Hybrid Computational Intelligence*, ser. Hybrid Computational Intelligence for Pattern Analysis and Understanding, S. Bhattacharyya, V. Snášel, D. Gupta, and A. Khanna, Eds. Academic Press, 2020, pp. 25–47. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128186992000020

[5] B. Balusamy, N. R, S. Kadry, and A. Gandomi, *Big Data: Concepts, Technology, and Architecture*. 111 River Street, Hoboken, NJ 07030, United States: Wiley, 2021, pp. 1–2. [Online]. Available: https://books.google.de/books?id=u7IfEAAAQBAJ

[6] ——, *Big Data: Concepts, Technology, and Architecture*. 111 River Street, Hoboken, NJ 07030, United States: Wiley, 2021, pp. 166–167. [Online]. Available: https://books.google.de/books?id=u7IfEAAAQBAJ

[7] A. Raj, J. Bosch, H. H. Olsson, and T. J. Wang, "Modelling data pipelines," in *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 2020, pp. 13–20.

[8] J. Densmore, *Data Pipelines Pocket Reference*, 1st ed. 1005 Gravenstein Highway North, Sebastopol, California CA 95472, United States: O'Reilly Media, 2021, pp. 17–19. [Online]. Available: https://books.google.de/books?id=SxgcEAAAQBAJ

[9] H. Hapke and C. Nelson, *Building Machine Learning Pipelines*, 1st ed. 1005 Gravenstein Highway North, Sebastopol, California CA 95472, United States: O'Reilly Media, 2020, pp. 8–10. [Online]. Available: https://books.google.de/books?id=H6_wDwAAQBAJ

[10] B. Harenslak and J. de Ruiter, *Data Pipelines with Apache Airflow*, 1st ed. 20 Baldwin Rd, Shelter Island, New York, 11964, United States: Manning Publications, 2021, pp. 9–11. [Online]. Available: https://books.google.com.tr/books?id=8EwnEAAAQBAJ

[11] R. J. Wilson, *Introduction to Graph Theory*, 4th ed. Edinburgh Gate, Harlow, Essex CM20 2JE, England: Longman, 1996, pp. 1–4. [Online]. Available: https://www.maths.ed.ac.uk/~v1ranick/papers/wilsongraph.pdf

[12] F. Harary, *Graph Theory*, 4th ed. Phillipines: Addison-Wesley Publishing Company, 1969, pp. 198–202. [Online]. Available: https://users.metu.edu.tr/aldoks/341/Book%201%20(Harary).pdf

[13] J. Densmore, *Data Pipelines Pocket Reference*, 1st ed. 1005 Gravenstein Highway North, Sebastopol, California CA 95472, United States: O'Reilly Media, 2021, pp. 1–5. [Online]. Available: https://books.google.de/books?id=SxgcEAAAQBAJ

[14] A. Silberschatz, P. B. Galvin, and G. Gagne, *Operating System Concepts*, 10th ed. 111 River Street, Hoboken, NJ 07030, United States: John Wiley and Sons, Inc., 2018, p. Glossary. [Online]. Available: https://os.ecci.ucr.ac.cr/slides/Abraham-Silberschatz-Operating-System-Concepts-10th-2018.pdf

[15] TechTarget. scalability. Accessed: 2023-05-07. [Online]. Available: https://www.techtarget.com/searchdatacenter/definition/scalability

[16] H. Liu, *Software Performance and Scalability: A Quantitative Approach*, ser. Quantitative Software Engineering Series. 111 River Street, Hoboken, NJ 07030, United States: John Wiley and Sons, Inc., 2009. [Online]. Available: https://download.e-bookshelf.de/download/0000/5732/96/L-G-0000573296-0002358512.pdf

[17] P. A. Laplante, *Dictionary of Computer Science, Engineering and Technology*, 1st ed. 2000 N.W. Corporate Blvd., Boca Raton, Florida 33431, United States: CRC Press LLC, 2001.

[18] M. Rouse. Modularity. Accessed: 2023-05-07. [Online]. Available: https://www.techopedia.com/definition/24772/modularity

[19] L. Shen and S. Ren, "Analysis and measurement of software flexibility based on flexible points," in *3rd Software Measurement European Forum, SMEF 2006. Proceedings.* Software Measurement European Forum, 2006, pp. 331–341.

[20] H. Mykhailyshyn, N. Pasyeka, V. Sheketa, M. Pasyeka, O. Kondur, and M. Varvaruk, "Designing network computing systems for intensive processing of information flows of data," in *Data-Centric Business and Applications: ICT Systems-Theory, Radio-Electronics, Information Technologies and Cybersecurity (Volume 5)*, ser. Lecture Notes on Data Engineering and Communications Technologies, T. Radivilova, D. Ageyev, and N. Kryvinska, Eds. Springer International Publishing, 2020, pp. 391–423. [Online]. Available: https://books.google.de/books?id=hHXsDwAAQBAJ

[21] S. Ligus, *Effective Monitoring and Alerting*, 1st ed. 1005 Gravenstein Highway North, Sebastopol, California CA 95472, United States: O'Reilly Media, 2012, pp. 1–3. [Online]. Available: https://books.google.de/books?id=KirJSqFcWIEC

[22] Cambridge Dictionary. efficient. Accessed: 2023-05-08. [Online]. Available: https://dictionary.cambridge.org/dictionary/english/efficient

[23] ——. effectively. Accessed: 2023-05-08. [Online]. Available: https://dictionary.cambridge.org/dictionary/english/effectively

[24] S. Haines, *Workflow Orchestration with Apache Airflow.* Berkeley, CA: Apress, 2022, pp. 255–295. [Online]. Available: https://doi.org/10.1007/978-1-4842-7452-1_8

[25] Kubernetes. kubernetes. Accessed: 2023-05-09. [Online]. Available: https://kubernetes.io/

[26] L. Finnigan and E. Toner, "Building and maintaining metadata aggregation workflows using apache airflow," *Code4Lib, Iss. 52*, 2021. [Online]. Available: https://journal.code4lib.org/articles/16171

[27] M. Hilgendorf, "Efficient industrial big data pipeline for lossless transfer of vehicular data," Master's thesis, Chalmers University of Technology, University of Gothenburg, 2022.

[28] R. Mitchell, L. Pottier, S. Jacobs, R. F. d. Silva, M. Rynge, K. Vahi, and E. Deelman, "Exploration of workflow management systems emerging features from users perspectives," in *2019 IEEE International Conference on Big Data (Big Data)*, 2019, pp. 4537–4544.

[29] National Institute of Standards and Technology (NIST). optimization problem. Accessed: 2023-05-08. [Online]. Available: https://xlinux.nist.gov/dads/HTML/optimization.html

[30] P. Kortelainen, "Panu kortelainen: Manage your workflows: A classification framework and technology review of workflow management systems," Master's thesis, Tampere University, 2021.

[31] WasteAnt. Wasteant. Accessed: 2023-05-09. [Online]. Available: https://wasteant.com/

[32] ROS. Bags. Accessed: 2023-05-09. [Online]. Available: http://wiki.ros.org/Bags

[33] ——. Publishers and subscribers. Accessed: 2023-05-09. [Online]. Available: http://wiki.ros.org/roscpp/Overview/Publishers%20and%20Subscribers

[34] ——. Documentation. Accessed: 2023-05-10. [Online]. Available: http://wiki.ros.org/

[35] ——. Nodes. Accessed: 2023-05-10. [Online]. Available: http://wiki.ros.org/Nodes

[36] ——. Topics. Accessed: 2023-05-10. [Online]. Available: http://wiki.ros.org/Topics

[37] Apache Software Foundation. What is airflow? Accessed: 2023-05-10. [Online]. Available: https://airflow.apache.org/docs/apache-airflow/stable/index.html

[38] Spotify. Luigi. Accessed: 2023-05-10. [Online]. Available: https://luigi.readthedocs.io/en/stable/

[39] Kubeflow. Documentation. Accessed: 2023-05-10. [Online]. Available: https://www.kubeflow.org/docs/

[40] Argo Workflows. Documentation. Accessed: 2023-05-10. [Online]. Available: https://argoproj.github.io/argo-workflows/

[41] PrefectHQ. Welcome to prefect. Accessed: 2023-05-10. [Online]. Available: https://docs.prefect.io/latest/

[42] Pachyderm. Pachyderm docs. Accessed: 2023-05-10. [Online]. Available: https://docs.pachyderm.com/

[43] IEEE Standards Association. Ieee draft standard for information technology–portable operating system interface (posix(tm)) base specifications, issue 8. Accessed: 2023-05-10. [Online]. Available: https://standards.ieee.org/ieee/1003.1/7700/#Standard

[44] R. M. Stallman. The origin of the name posix. Accessed: 2023-05-10. [Online]. Available: https://stallman.org/articles/posix.html

[45] Apache Software Foundation. What is airflow? Accessed: 2023-05-10. [Online]. Available: https://airflow.apache.org/docs/apache-airflow/stable/index.html

[46] Docker. Develop faster. run anywhere. Accessed: 2023-05-11. [Online]. Available: https://www.docker.com/

[47] E. S. Raymond, *The Cathedral and The Bazaar*. Texas, USA: Snowball Publishing, 2010, pp. 7–10.

[48] Apache Software Foundation. Community. Accessed: 2023-05-11. [Online]. Available: https://airflow.apache.org/community/

[49] Spotify. Issues. Accessed: 2023-05-11. [Online]. Available: https://github.com/spotify/luigi/issues

[50] PrefectHQ. Community. Accessed: 2023-05-11. [Online]. Available: https://www.prefect.io/community/

[51] Google. Basics of google trends. Accessed: 2023-05-10. [Online]. Available: https://newsinitiative.withgoogle.com/resources/lessons/basics-of-google-trends/

[52] star-history.com. Star history. Accessed: 2023-04-20. [Online]. Available: https://star-history.com/#apache/airflow&spotify/luigi&PrefectHQ/prefect&Date

[53] Google Trends. apache airflow spotify lugi prefecthq prefect. Accessed: 2023-04-20. [Online]. Available: https://trends.google.com/trends/explore?cat=5&date=2012-09-20%202023-04-20&q=apache%20airflow,spotify%20luigi,prefecthq%20prefect&hl=en

[54] Stack Exchange. Stack exchange data explorer. Accessed: 2023-04-20. [Online]. Available: https://data.stackexchange.com/help

[55] M. Rouse. Production environment. Accessed: 2023-05-11. [Online]. Available: https://www.techopedia.com/definition/8989/production-environment

[56] Apache Software Foundation. Dags. Accessed: 2023-05-11. [Online]. Available: https://airflow.apache.org/docs/apache-airflow/stable/core-concepts/dags.html

[57] ——. Tasks. Accessed: 2023-05-11. [Online]. Available: https://airflow.apache.org/docs/apache-airflow/stable/core-concepts/tasks.html

[58] ——. PythonOperator. Accessed: 2023-05-11. [Online]. Available: https://airflow.apache.org/docs/apache-airflow/stable/howto/operator/python.html

[59] ——. Architecture overview. Accessed: 2023-05-11. [Online]. Available: https://airflow.apache.org/docs/apache-airflow/stable/core-concepts/overview.html

[60] Celery. Introduction to celery. Accessed: 2023-05-11. [Online]. Available: https://docs.celeryq.dev/en/latest/getting-started/introduction.html

[61] GitLab. What is ci/cd? Accessed: 2023-05-12. [Online]. Available: https://about.gitlab.com/topics/ci-cd/#ci-cd-explained

[62] Helm. The package manager for Kubernetes. Accessed: 2023-05-12. [Online]. Available: https://helm.sh/

[63] J. Helgeson, *The Software Audit Guide*. ASQ, P.O. Box 3005, Milwaukee, WI 53201-3005, United States: ASQ Quality Press, 2009, p. 32. [Online]. Available: https://books.google.de/books?id=9OqiEAAAQBAJ

[64] D. Curran-Everett, S. Taylor, and K. Kafadar, "Fundamental concepts in statistics: elucidation and illustration," *Journal of Applied Physiology*, vol. 85, no. 3, pp. 775–786, 1998.

[65] D. K. Lee, J. In, and S. Lee, "Standard deviation and standard error of the mean," *Korean Journal of Anesthesiology*, vol. 68, no. 3, pp. 220–223, 2015.

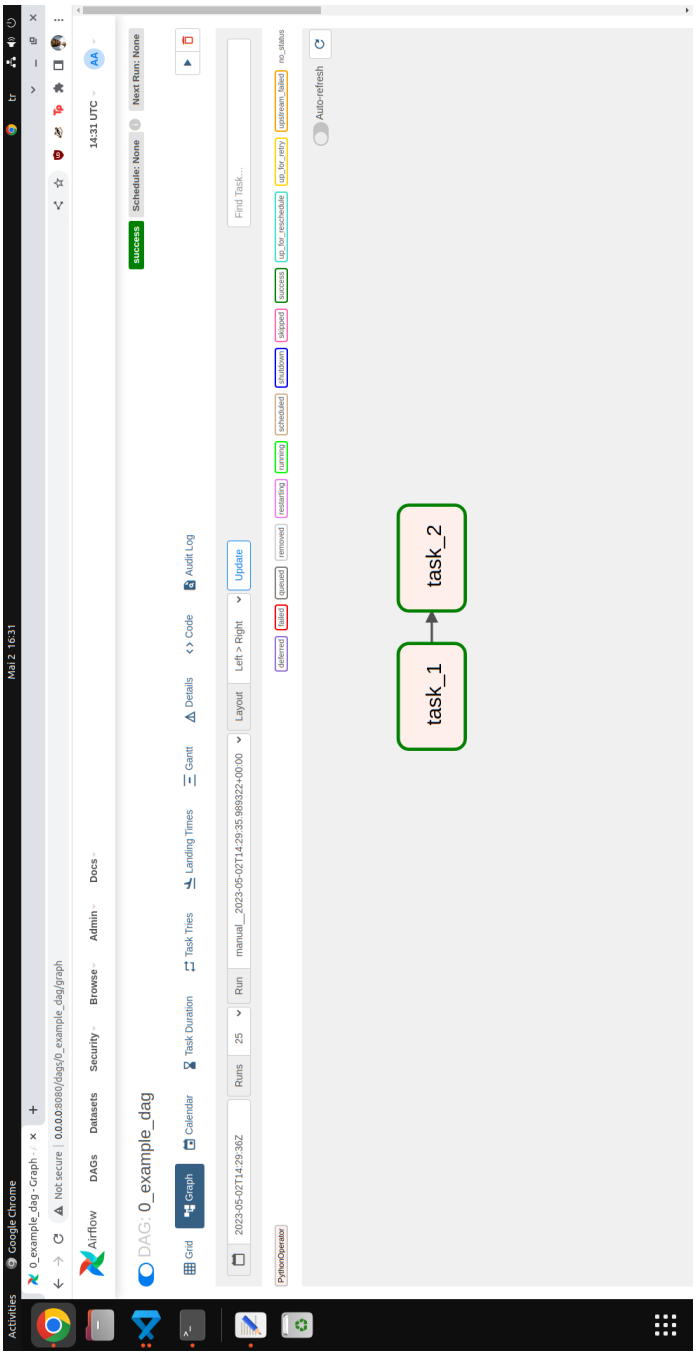
[66] N. Jones and P. Pevzner, *An Introduction to Bioinformatics Algorithms*, ser. Computational Molecular Biology. One Broadway 12th Floor Cambridge, MA, United states: MIT Press, 2004, pp. 37–38. [Online]. Available: https://books.google.de/books?id=p_qzpkNVcUwC

Appendix A

Category		Details
Project view	Licensing	Used licenses
		Limitations
	Installation	Platform
		Dependencies
		Docker image availability
	Source Code and Maturity	Programming Language
		Stage of Maturity
		Project Age
	Community	Community Page Availability
		Github Stars, Contributors, Releases
		Stack Overflow questions
		Google Trends Interest Score
	Interface Availability	CLI
		API
		GUI
	Documentation	Workflow Development
		Production Deployment
		Architecture
		Platform Development
Technical view	Development	Workflow Definition
		Task Implementation
		IDE Extensions
	Architecture	High Availability
		Scalability
		State Persistency
		Asynchronous Messaging
	Workflows	Feature Support
		Control Mechanisms
	Application Delivery	Deployment Automation
		CI/CD Utilities
	Code Reuse	Workflow Examples
		Code Samples
	Available Integrations	Cloud Environments
		External Software Assets

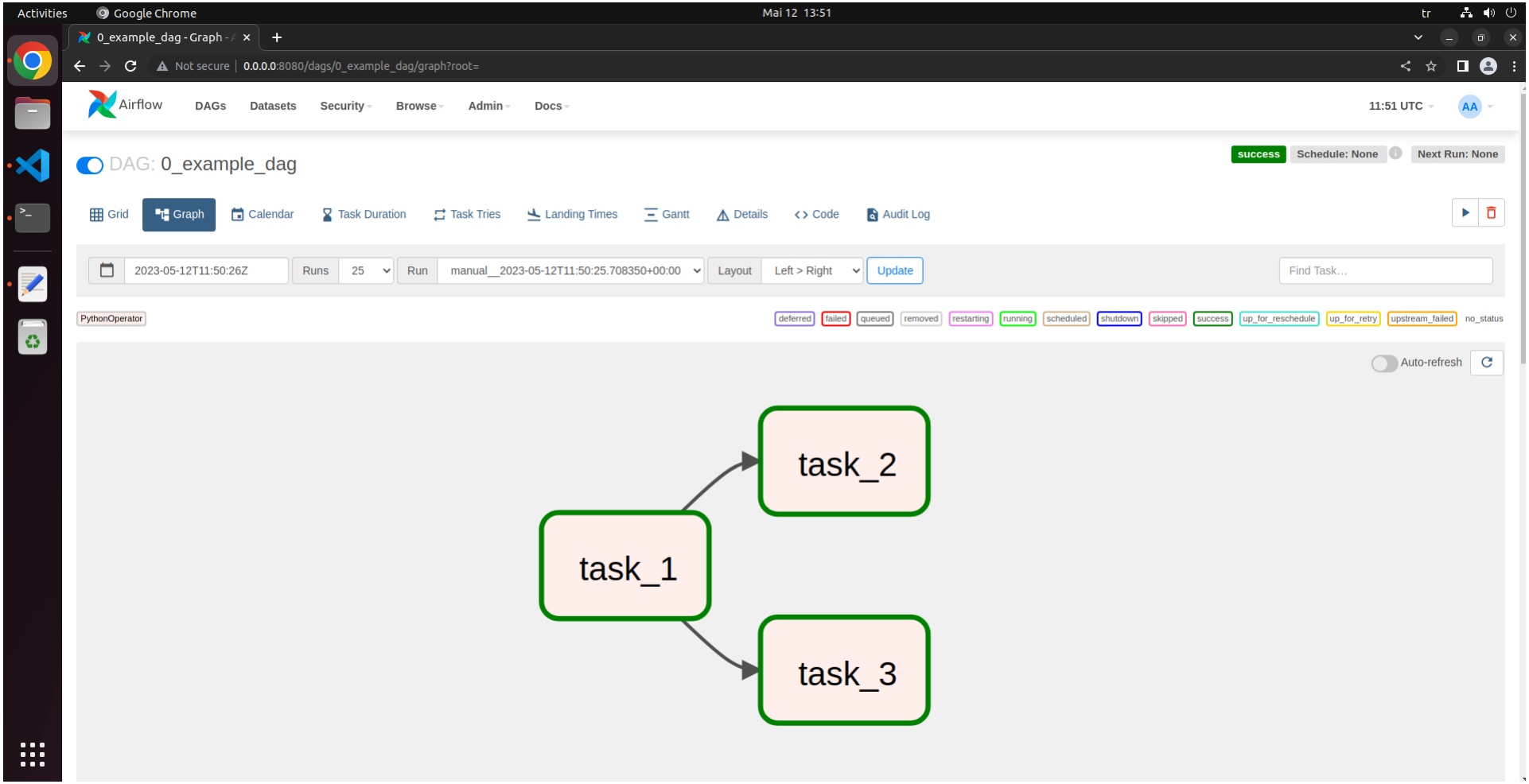
WMS Selection Criteria obtained from Kortelainen [30]. Google Trends Interest Score is a replacement for Google Hits. Reason discussed on Section 4.1.2.

Appendix B


Activities
Google Chrome
Mai 2 16:31
0_example_dag - Graph
Not secure | 0.0.0.0:8080/dags/0_example_dag/graph
Airflow
DAGs
Datasets
Security
Browse
Admin
Docs
14:31 UTC
AA
DAG: 0_example_dag
success
Schedule: None
Next Run: None
Grid
Graph
Calendar
Task Duration
Task Tries
Landing Times
Gantt
Details
Code
Audit Log
2023-05-02T14:29:36Z
Runs
25
Run
manual__2023-05-02T14:29:35.989322+00:00
Layout
Left > Right
Update
Find Task...
Auto-refresh
PythonOperator
deferred
failed
queued
removed
restarting
running
scheduled
shutdown
skipped
success
up_for_reschedule
up_for_retry
upstream_failed
no_status
task_1
task_2

```
1  from airflow import DAG
2  from airflow.operators.python import PythonOperator
3
4  from datetime import datetime
5
6  default_args = {
7      'owner': 'airflow',
8      'depends_on_past': False,
9      'start_date': datetime(2023, 4, 10),
10     'retries': 0
11 }
12
13 dag = DAG(
14     dag_id='0_example_dag',
15     default_args=default_args,
16     schedule_interval=None,
17     max_active_runs=1
18 )
19
20 def task(x: int):
21     print(x)
22
23 task_1 = PythonOperator(
24     task_id='task_1',
25     python_callable=task,
26     op_kwargs={'x': 1},
27     dag=dag,
28 )
29
30 task_2 = PythonOperator(
31     task_id='task_2',
32     python_callable=task,
33     op_kwargs={'x': 2},
34     dag=dag,
35 )
36
37 task_1 >> task_2
```

A DAG with two Python Operators executing the same function. Figure displays the appearance on Airflow Web-UI and code snippet shows the Python code.


Activities
Google Chrome
Mai 12 13:51
tr
0_example_dag - Graph -
Not secure
0.0.0.0:8080/dags/0_example_dag/graph?root=
Airflow
DAGs
Datasets
Security
Browse
Admin
Docs
11:51 UTC
AA
DAG: 0_example_dag
success
Schedule: None
Next Run: None
Grid
Graph
Calendar
Task Duration
Task Tries
Landing Times
Gantt
Details
Code
Audit Log
2023-05-12T11:50:26Z
Runs
25
Run
manual__2023-05-12T11:50:25.708350+00:00
Layout
Left > Right
Update
Find Task...
PythonOperator
deferred
failed
queued
removed
restarting
running
scheduled
shutdown
skipped
success
up_for_reschedule
up_for_retry
upstream_failed
no_status
Auto-refresh
task_1
task_2
task_3

```
from airflow import DAG
from airflow.operators.python import PythonOperator

...

task_1 = PythonOperator(
    task_id='task_1',
    python_callable=task,
    op_kwargs={'x': 1},
    dag=dag,
)

task_2 = PythonOperator(
    task_id='task_2',
    python_callable=task,
    op_kwargs={'x': 2},
    dag=dag,
)

task_3 = PythonOperator(
    task_id='task_3',
    python_callable=task,
    op_kwargs={
        'x': 1
    },
    dag=dag,
)

task_1 >> [task_2, task_3]
```

Scaled-up version of the initial implementation with the addition of a third operator.

Appendix C

```
1  ...
2
3  [celery]
4
5  # This section only applies if you are using the CeleryExecutor in
6  # ``[core]`` section above
7  # The app name that will be used by celery
8  celery_app_name = airflow.executors.celery_executor
9
10 # The concurrency that will be used when starting workers with the
11 # ``airflow celery worker`` command. This defines the number of task
       instances that
12 # a worker will take, so size up your workers based on the resources
       on
13 # your worker box and the nature of your tasks
14 worker_concurrency = 16
15
16 ...
```

An extract from Airflow's configuration file, detailing how many concurrent thread that a Celery worker could have. Concurrency means that a worker can take in charge of a multitude of tasks.

Appendix D

```
1  import os
2  import random
3  import time
4  from datetime import datetime, timedelta
5
6  from airflow import DAG
7  from airflow.operators.python import PythonOperator
8
9  from logger import log_line
10
11 log_name = "large_dag_airflow_execution"
12 LOG_FILE = os.getcwd() + "/dags/log/"+log_name+".log"
13
14 default_args = {
15     'owner': 'airflow',
16     'depends_on_past': False,
17     'start_date': datetime(2023, 4, 10),
18     'retries': 0
19 }
20
21 dag = DAG(
22     dag_id='0_large_dag',
23     default_args=default_args,
24     schedule_interval=None,
25     max_active_runs=1
26 )
27
28 ####################################
29 ##### Configuration Variables #####
30 ####################################
```

```
31
32
33 seconds = 1
34 throughput_measure_intervals = [
35     (15 * seconds),
36     (30 * seconds),
37     (45 * seconds),
38     ((1 * 60) * seconds),
39     (((1 * 60) * seconds) + (15 * seconds)),
40     (((1 * 60) * seconds) + (30 * seconds)),
41     (((1 * 60) * seconds) + (45 * seconds)),
42     ((2 * 60) * seconds),
43     (((2 * 60) * seconds) + (15 * seconds)),
44     (((2 * 60) * seconds) + (30 * seconds)),
45     (((2 * 60) * seconds) + (45 * seconds)),
46     ((3 * 60) * seconds),
47     (((3 * 60) * seconds) + (15 * seconds)),
48     (((3 * 60) * seconds) + (30 * seconds)),
49     (((3 * 60) * seconds) + (45 * seconds)),
50     ((4 * 60) * seconds),
51     (((4 * 60) * seconds) + (15 * seconds)),
52     (((4 * 60) * seconds) + (30 * seconds)),
53     (((4 * 60) * seconds) + (45 * seconds)),
54     ((5 * 60) * seconds),
55     (((5 * 60) * seconds) + (15 * seconds)),
56     (((5 * 60) * seconds) + (30 * seconds)),
57     (((5 * 60) * seconds) + (45 * seconds)),
58     ((6 * 60) * seconds),
59     (((6 * 60) * seconds) + (15 * seconds)),
60     (((6 * 60) * seconds) + (30 * seconds)),
61     (((6 * 60) * seconds) + (45 * seconds)),
62     ((7 * 60) * seconds),
63     (((7 * 60) * seconds) + (15 * seconds)),
64     (((7 * 60) * seconds) + (30 * seconds)),
65     (((7 * 60) * seconds) + (45 * seconds)),
66     ((8 * 60) * seconds),
67 ]
68
69 def get_extract_transform_params():
```

```
70      #random.seed(seed)
71      runtime = random.randint(10, 15) * seconds
72      input = random.randint(800, 1200)
73      return [runtime, input]
74
75  def get_processor_params():
76      #random.seed(seed)
77      runtime = random.randint(25, 30) * seconds
78      input = random.randint(600, 750)
79      return [runtime, input]
80
81  def get_classifier_params():
82      #random.seed(seed)
83      runtime = random.randint(40, 45) * seconds
84      input = random.randint(400, 600)
85      return [runtime, input]
86
87  ############################
88  ##### Mimic Functions #####
89  ############################
90
91  def set_start(**context):
92      #random.seed(111)
93      dag_start = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
94      context['ti'].xcom_push(key='dag_start', value=dag_start)
95
96  def mimic(name, kernel, **context):
97      runtime, input = kernel()
98      time.sleep(runtime)
99
100     dag_start = datetime.strptime(context['ti'].xcom_pull(task_ids='
            start_task', key='dag_start'), "%Y-%m-%d %H:%M:%S")
101     end = datetime.now()
102     for index, value in enumerate(throughput_measure_intervals):
103         match_flag = False
104         if index == 0:
105             if (end - dag_start) <= timedelta(seconds=value):
106                 context['ti'].xcom_push(key=f'{name}_throughput',
                        value=input)
```

```
107                 context['ti'].xcom_push(key=f'{name}_record_time',
                        value=value)
108                 match_flag = True
109         else:
110             if (end - dag_start) > timedelta(seconds=
                    throughput_measure_intervals[index-1]) and (end -
                    dag_start) <= timedelta(seconds=value):
111                 context['ti'].xcom_push(key=f'{name}_throughput',
                        value=input)
112                 context['ti'].xcom_push(key=f'{name}_record_time',
                        value=value)
113                 match_flag = True
114         if match_flag:
115             break
116
117 def print_total_throughput(**context):
118     start = datetime.now()
119     dag_start = datetime.strptime(context['ti'].xcom_pull(task_ids='
            start_task', key='dag_start'), "%Y-%m-%d %H:%M:%S")
120     delta = start - dag_start
121     log_line(LOG_FILE, "------ START LOG ------")
122     log_line(LOG_FILE, "Total Runtime: " + str(delta))
123
124     task_list = []
125     for task in ['extract_transform_1', 'extract_transform_2', '
            extract_transform_3', 'processor_1', 'processor_2', '
            processor_3', 'processor_4', 'processor_5', 'processor_6', '
            classifier_1', 'classifier_2', 'classifier_3', 'classifier_4'
            ]:
126         throughput = context['ti'].xcom_pull(task_ids=task, key=f'{
                task}_throughput')
127         record_time = context['ti'].xcom_pull(task_ids=task, key=f'{
                task}_record_time')
128
129         if throughput and record_time:
130             #total_throughput += throughput
131
132             if "extract" in task:
133                 task_list.append(
```

```
134                     {
135                         'task' : task,
136                         'level': 1,
137                         'record_time': record_time,
138                         'throughput': throughput
139                     }
140                 )
141 
142             if "processor" in task:
143                 task_list.append(
144                     {
145                         'task' : task,
146                         'level': 2,
147                         'record_time': record_time,
148                         'throughput': throughput
149                     }
150                 )
151 
152             if "classifier" in task:
153                 task_list.append(
154                     {
155                         'task' : task,
156                         'level': 3,
157                         'record_time': record_time,
158                         'throughput': throughput
159                     }
160                 )
161             #log_line(LOG_FILE, "Throughput: " + str(
                    total_throughput) + " record time: " + str(
                    record_time) +" task: " + task + " throughput: " +
                    str(throughput))
162     def sort_key(item):
163         return (item["level"], item["record_time"])
164 
165     task_list = sorted(task_list, key=sort_key)
166     log_line(LOG_FILE, str(task_list))
167     total_throughput = 0
168     for task in task_list:
169         total_throughput += task['throughput']
```

```
170             log_line(LOG_FILE, "Throughput: " + str(total_throughput) +
                    " record time: " + str(task['record_time']) +" task: " +
                    str(task['task']) + " throughput: " + str(task['
                    throughput']))
171
172         log_line(LOG_FILE, "Total throughput: " + str(total_throughput))
173
174         log_line(LOG_FILE, "------- END LOG -------")
175
176 ########################################
177 ##### Extract Transform Operators #####
178 ########################################
179
180 start_task = PythonOperator(
181     task_id = 'start_task',
182     python_callable=set_start,
183     dag=dag
184
185 )
186
187 extract_transform_1 = PythonOperator(
188     task_id='extract_transform_1',
189     python_callable=mimic,
190     op_kwargs={
191         'name': 'extract_transform_1',
192         'kernel': get_extract_transform_params
193     },
194     dag=dag,
195 )
196
197 extract_transform_2 = PythonOperator(
198     task_id='extract_transform_2',
199     python_callable=mimic,
200     op_kwargs={
201         'name': 'extract_transform_2',
202         'kernel': get_extract_transform_params
203     },
204     dag=dag,
205 )
```

```
206
207 extract_transform_3 = PythonOperator(
208     task_id='extract_transform_3',
209     python_callable=mimic,
210     op_kwargs={
211         'name': 'extract_transform_3',
212         'kernel': get_extract_transform_params
213     },
214     dag=dag,
215 )
216
217 ################################
218 ##### Processor Operators #####
219 ################################
220
221 processor_1 = PythonOperator(
222     task_id='processor_1',
223     python_callable=mimic,
224     op_kwargs={
225         'name': 'processor_1',
226         'kernel': get_processor_params
227     },
228     dag=dag,
229 )
230
231 processor_2 = PythonOperator(
232     task_id='processor_2',
233     python_callable=mimic,
234     op_kwargs={
235         'name': 'processor_2',
236         'kernel': get_processor_params
237     },
238     dag=dag,
239 )
240
241 processor_3 = PythonOperator(
242     task_id='processor_3',
243     python_callable=mimic,
244     op_kwargs={
```

```
245             'name': 'processor_3',
246             'kernel': get_processor_params
247         },
248         dag=dag,
249     )
250
251     processor_4 = PythonOperator(
252         task_id='processor_4',
253         python_callable=mimic,
254         op_kwargs={
255             'name': 'processor_4',
256             'kernel': get_processor_params
257         },
258         dag=dag,
259     )
260
261     processor_5 = PythonOperator(
262         task_id='processor_5',
263         python_callable=mimic,
264         op_kwargs={
265             'name': 'processor_5',
266             'kernel': get_processor_params
267         },
268         dag=dag,
269     )
270
271     processor_6 = PythonOperator(
272         task_id='processor_6',
273         python_callable=mimic,
274         op_kwargs={
275             'name': 'processor_6',
276             'kernel': get_processor_params
277         },
278         dag=dag,
279     )
280
281     ##################################
282     ##### Classifier Operators #####
283     ##################################
```

```
284
285 classifier_1 = PythonOperator(
286     task_id='classifier_1',
287     python_callable=mimic,
288     op_kwargs={
289         'name': 'classifier_1',
290         'kernel': get_classifier_params
291     },
292     dag=dag,
293 )
294
295 classifier_2 = PythonOperator(
296     task_id='classifier_2',
297     python_callable=mimic,
298     op_kwargs={
299         'name': 'classifier_2',
300         'kernel': get_classifier_params
301     },
302     dag=dag,
303 )
304
305 classifier_3 = PythonOperator(
306     task_id='classifier_3',
307     python_callable=mimic,
308     op_kwargs={
309         'name': 'classifier_3',
310         'kernel': get_classifier_params
311     },
312     dag=dag,
313 )
314
315 classifier_4 = PythonOperator(
316     task_id='classifier_4',
317     python_callable=mimic,
318     op_kwargs={
319         'name': 'classifier_4',
320         'kernel': get_classifier_params
321     },
322     dag=dag,
```

```
)

end_task = PythonOperator(
    task_id='end_task',
    python_callable=print_total_throughput,
    dag=dag,
)

start_task >> [extract_transform_1, extract_transform_2,
    extract_transform_3]
extract_transform_1 >> [processor_1, processor_2, processor_3] >>
    classifier_1
extract_transform_2 >> [processor_2, processor_3, processor_4,
    processor_5]
[processor_2, processor_3, processor_4] >> classifier_2
[processor_1, processor_2, processor_3, processor_4] >> classifier_3
extract_transform_3 >> [processor_5, processor_6] >> classifier_4
[classifier_1, classifier_2, classifier_3, classifier_4] >> end_task
```

Code for large DAG used in implementation and evaluation, code is also available on GitHub `https://github.com/canyetismis/20230515-DAGS`, alongside with the codes of other implementations

Appendix E

	Trials										
Time Intervals (Time Intervals (	1	2	3	4	5	6	7	8	9	10	Average
15 0:15	1090	0	0	0	0	0	0	0	1135	0	222.5
30 0:30	2206	1938	2050	1026	2349	957	1903	1716	1135	1879	1715.9
45 0:45	2206	1938	2050	1904	2349	1977	1903	1716	2189	1879	2011.1
60 1:00	2899	2637	2789	1904	2349	1977	2521	2386	2901	2626	2498.9
75 1:15	2899	2637	2789	2550	3026	2602	2521	2386	2901	2626	2693.7
90 1:30	3504	2637	3437	3223	3026	3312	3227	3108	3538	3313	3232.5
105 1:45	3504	3287	3437	3223	3758	3312	3227	3108	3538	3313	3370.7
120 2:00	4190	3287	4155	3872	3758	3964	3858	3809	4256	4017	3916.6
135 2:15	4190	3896	4155	3872	4428	3964	3858	3809	4256	4017	4044.5
150 2:30	4891	4646	4788	4598	4428	4606	4468	4426	4903	4704	4645.8
165 2:45	4891	4646	4788	4598	5101	4606	4468	4426	4903	4704	4713.1
180 3:00	4891	4646	4788	4598	5101	4606	4468	4426	4903	4704	4713.1
195 3:15	5422	5047	5226	5106	5101	5044	4938	4972	5333	5254	5144.3
210 3:30	5422	5047	5226	5106	5508	5044	4938	4972	5333	5254	5185
225 3:45	5422	5047	5226	5106	5508	5044	4938	4972	5333	5254	5185
240 4:00	5917	5499	5786	5662	5508	5581	5499	5393	5752	5837	5643.4
255 4:15	5917	5499	5786	5662	6031	5581	5499	5393	5752	5837	5695.7
270 4:30	5917	5499	5786	5662	6031	5581	5499	5393	5752	5837	5695.7
285 4:45	6318	6026	6370	6120	6031	6003	5946	5993	6258	6336	6140.1
300 5:00	6318	6026	6370	6120	6554	6003	5946	5993	6258	6336	6192.4
Total Data	6318	6026	6370	6120	6554	6003	5946	5993	6258	6336	6192.4
Total Runtime	0:04:41	0:04:46	0:04:39	0:04:47	0:04:55	0:04:44	0:04:45	0:04:46	0:04:46	0:04:41	0:04:45

Raw data for sequential execution of small DAG

		Trials										
Time Intervals (s)	Time Intervals (	1	2	3	4	5	6	7	8	9	10	Average
15	0:15	1859	811	0	968	1096	0	1872	2036	1712	1021	1137.5
30	0:30	1859	1830	1661	2159	2255	2003	1872	2036	1712	2026	1941.3
45	0:45	4003	3125	2964	2901	4300	4152	3298	3393	4527	3419	3608.2
60	1:00	4630	4456	4319	4928	5041	4868	4555	4819	4527	4639	4678.2
75	1:15	4630	4456	4319	4928	5041	4868	4555	4819	4527	4639	4678.2
90	1:30	4630	4922	4319	4928	5992	4868	4555	4819	5538	4639	4921
105	1:45	6124	5874	5798	6352	6403	6448	6210	6470	6036	6101	6181.6
120	2:00	6124	5874	5798	6352	6403	6448	6210	6470	6036	6101	6181.6
135	2:15	6124	5874	5798	6352	6403	6448	6210	6470	6036	6101	6181.6
150	2:30	6124	5874	5798	6352	6403	6448	6210	6470	6036	6101	6181.6
165	2:45	6124	5874	5798	6352	6403	6448	6210	6470	6036	6101	6181.6
180	3:00	6124	5874	5798	6352	6403	6448	6210	6470	6036	6101	6181.6
195	3:15	6124	5874	5798	6352	6403	6448	6210	6470	6036	6101	6181.6
210	3:30	6124	5874	5798	6352	6403	6448	6210	6470	6036	6101	6181.6
225	3:45	6124	5874	5798	6352	6403	6448	6210	6470	6036	6101	6181.6
240	4:00	6124	5874	5798	6352	6403	6448	6210	6470	6036	6101	6181.6
255	4:15	6124	5874	5798	6352	6403	6448	6210	6470	6036	6101	6181.6
270	4:30	6124	5874	5798	6352	6403	6448	6210	6470	6036	6101	6181.6
285	4:45	6124	5874	5798	6352	6403	6448	6210	6470	6036	6101	6181.6
300	5:00	6124	5874	5798	6352	6403	6448	6210	6470	6036	6101	6181.6
	Total Data	6124	5874	5798	6352	6403	6448	6210	6470	6036	6101	6181.6
	Total Runtime	0:01:33	0:01:35	0:01:37	0:01:35	0:01:33	0:01:37	0:01:34	0:01:35	0:01:32	0:01:36	0:01:35

Raw data for Airflow execution of small DAG

		Trials										
Time Intervals (	Time Intervals (	1	2	3	4	5	6	7	8	9	10	Average
15	0:15	1198	964	0	1150	812	0	952	0	0	1155	623.1
30	0:30	2156	2108	921	1150	1651	1018	952	907	1040	2177	1408
45	0:45	3348	3226	1977	2248	2592	2178	2847	1775	1851	2177	2421.9
60	1:00	3348	3226	3133	3111	2592	3098	2847	2948	2880	3235	3041.8
75	1:15	4011	3909	3133	3111	3215	3098	3505	2948	2880	3235	3304.5
90	1:30	4011	3909	3736	3834	3215	3792	3505	3680	3590	3870	3714.2
105	1:45	4734	3909	3736	3834	3215	3792	4175	3680	3590	3870	3853.5
120	2:00	4734	4579	4385	4470	3820	4456	4175	4296	4219	4592	4372.6
135	2:15	5472	4579	5044	4470	3820	4456	4872	4296	4219	5317	4654.5
150	2:30	5472	5235	5044	5128	4515	5183	4872	4924	4960	5317	5065
165	2:45	6145	5235	5728	5128	4515	5183	5529	4924	4960	6001	5334.8
180	3:00	6145	5944	5728	5852	5167	5811	5529	5542	5629	6001	5734.8
195	3:15	6775	5944	6442	5852	5167	5811	6207	5542	5629	6627	5999.6
210	3:30	6775	6664	6442	6576	5869	6492	6207	6273	6336	6627	6426.1
225	3:45	7467	6664	7115	6576	5869	6492	6901	7000	6336	7328	6774.8
240	4:00	7467	7305	7115	7210	6472	7102	6901	7000	7048	7328	7094.8
255	4:15	7467	7305	7115	7210	6472	7102	6901	7000	7048	7328	7094.8
270	4:30	7889	7305	7115	7210	6472	7102	7406	7537	7048	7915	7299.9
285	4:45	7889	7720	7685	7711	7026	7653	7406	7537	7597	7915	7613.9
300	5:00	7889	7720	7685	7711	7026	7653	7406	7537	7597	7915	7613.9
315	5:15	8396	7720	8109	7711	7026	7653	7957	8037	7597	8431	7863.7
330	5:30	8396	8251	8109	8311	7476	8153	7957	8037	8055	8431	8117.6
345	5:45	8906	8251	8109	8311	7476	8153	7957	8037	8055	8431	8168.6
360	6:00	8906	8251	8109	8311	7476	8153	8475	8517	8055	8929	8318.2
375	6:15	8906	8791	8534	8889	7881	8561	8475	8517	8594	8929	8607.7
390	6:30	8906	8791	8534	8889	7881	8561	8475	8517	8594	8929	8607.7
405	6:45	9329	8791	8992	8889	7881	8561	8994	8953	8594	9448	8843.2
420	7:00	9329	9278	8992	9452	8375	9004	8994	8953	9161	9448	9098.6
435	7:15	9329	9278	8992	9452	8375	9004	8994	8953	9161	9448	9098.6
450	7:30	9329	9278	8992	9452	8375	9004	8994	8953	9161	9448	9098.6
465	7:45	9329	9278	8992	9452	8375	9004	8994	8953	9161	9448	9098.6
480	8:00	9329	9278	8992	9452	8375	9004	8994	8953	9161	9448	9098.6
	Total Data	9329	9278	8992	9452	8375	9004	8994	8953	9161	9448	9098.6
	Total Runtime	0:06:33	0:06:58	0:06:46	0:06:56	0:06:52	0:06:54	0:06:41	0:06:44	0:06:49	0:06:42	0:06:48

Raw data for sequential execution of large DAG

Raw data for Airflow execution of large DAG

		Trials										
Time Intervals (	Time Intervals (	1	2	3	4	5	6	7	8	9	10	Average
15	0:15	945	1012	807	841	2743	1003	969	3052	2284	1194	1485
30	0:30	3005	3083	2880	2712	2743	3272	2802	3052	3336	3353	3023.8
45	0:45	3646	4420	4915	4868	6799	4008	4209	7075	7579	5431	5295
60	1:00	7047	7151	6873	6964	6799	7309	6927	7075	7579	7555	7127.9
75	1:15	7047	7151	6873	6964	6799	7309	6927	7075	7579	7555	7127.9
90	1:30	7047	8210	7755	7811	8834	7887	6927	8955	9525	8443	8139.4
105	1:45	9325	9241	8685	8939	8834	9308	8824	8955	9525	9574	9121
120	2:00	9325	9241	8685	8939	8834	9308	8824	8955	9525	9574	9121
135	2:15	9325	9241	8685	8939	8834	9308	8824	8955	9525	9574	9121
150	2:30	9325	9241	8685	8939	8834	9308	8824	8955	9525	9574	9121
165	2:45	9325	9241	8685	8939	8834	9308	8824	8955	9525	9574	9121
180	3:00	9325	9241	8685	8939	8834	9308	8824	8955	9525	9574	9121
195	3:15	9325	9241	8685	8939	8834	9308	8824	8955	9525	9574	9121
210	3:30	9325	9241	8685	8939	8834	9308	8824	8955	9525	9574	9121
225	3:45	9325	9241	8685	8939	8834	9308	8824	8955	9525	9574	9121
240	4:00	9325	9241	8685	8939	8834	9308	8824	8955	9525	9574	9121
255	4:15	9325	9241	8685	8939	8834	9308	8824	8955	9525	9574	9121
270	4:30	9325	9241	8685	8939	8834	9308	8824	8955	9525	9574	9121
285	4:45	9325	9241	8685	8939	8834	9308	8824	8955	9525	9574	9121
300	5:00	9325	9241	8685	8939	8834	9308	8824	8955	9525	9574	9121
315	5:15	9325	9241	8685	8939	8834	9308	8824	8955	9525	9574	9121
330	5:30	9325	9241	8685	8939	8834	9308	8824	8955	9525	9574	9121
345	5:45	9325	9241	8685	8939	8834	9308	8824	8955	9525	9574	9121
360	6:00	9325	9241	8685	8939	8834	9308	8824	8955	9525	9574	9121
375	6:15	9325	9241	8685	8939	8834	9308	8824	8955	9525	9574	9121
390	6:30	9325	9241	8685	8939	8834	9308	8824	8955	9525	9574	9121
405	6:45	9325	9241	8685	8939	8834	9308	8824	8955	9525	9574	9121
420	7:00	9325	9241	8685	8939	8834	9308	8824	8955	9525	9574	9121
435	7:15	9325	9241	8685	8939	8834	9308	8824	8955	9525	9574	9121
450	7:30	9325	9241	8685	8939	8834	9308	8824	8955	9525	9574	9121
465	7:45	9325	9241	8685	8939	8834	9308	8824	8955	9525	9574	9121
480	8:00	9325	9241	8685	8939	8834	9308	8824	8955	9525	9574	9121
	Total Data	9325	9241	8685	8939	8834	9308	8824	8955	9525	9574	9121
	Total Runtime	0:01:34	0:01:33	0:01:35	0:01:34	0:01:31	0:01:37	0:01:37	0:01:31	0:01:32	0:01:36	0:01:34