POZNAN UNIVERSITY OF TECHNOLOGY

FACULTY OF COMPUTING AND TELECOMMUNICATION
Institute of Computing Science

IMPLEMENTATION AND ANALYSIS OF AIRBYTE CONNECTORS FOR APACHE ARROW FLIGHT

Radosław Burdziński, 141195
Huber Lindenau, 141077
Michał Szczepaniak, 141317
Filip Szóstak, 1413120

Supervisors
Assoc. Prof., PhD DSc. Robert Wrembel
Michał Bodziony, IBM

POZNAŃ 2022

Abstract

This report describes the research results on connectors to Airbyte for Apache Arrow Flight. As a first part of the research, the source and destination connectors were implemented as well as the Apache Arrow Flight server. Subsequently, the performance of developed connectors was analysed by making measurements on AWS and comparing results with the other connectors' performance.

Contents

Chapter 1

Introduction

The following chapter introduces the problem and requirements of the project. Additionally, it describes the main technologies used in the project - Airbyte and Apache Arrow Flight - along with their architectures.

1.1 Problem introduction

One of the most important tasks while working with data is extracting it from a source and afterwards loading it to a destination for further operations. The variety of possible sources and destinations makes it difficult to create a universal solution.

Airbyte is an open-source project whose purpose is to simplify the moving of data from source to destination. It offers a great number of source connectors which extract data from sources like SQL and NoSQL databases, Cloud Storages and many other systems. Simultaneously, it offers similar destination connectors which enable loading data to mentioned solutions. After configuring connectors, a user can establish a connection between a source and destination connectors to transfer data.

Despite the great number of available connectors, there are no connectors for communication with the Apache Arrow Flight server. Apache Arrow Flight is a part of the Apache Arrow framework dedicated to creating client-server architecture for transferring big amounts of data. Data sent by a client to the server can be then saved or used for further processing.

1.2 Requirements

The aim of the research was to analyse the efficiency of the Airbyte connectors enabling communication with Apache Arrow Flight servers.

For this reason, the main requirement was to develop the source and destination connectors. The connectors had to implement APIs enabling communication with Airbyte as well as with Apache Arrow Flight servers.

Additionally, to be able to compare the created connectors with existing ones, there was a need to conduct performance tests. The tests are required to be performed in a stable and reproducible environment to enable an analysis of their results.

1.3 Technologies

1.3.1 Airbyte

Airbyte overview

Airbyte is an open-source project whose purpose is to simplify the migrating of data between different databases and systems [1]. Its base elements are connectors, which enable communication with various services among which one can find:

- SQL databases - e.g. MySQL, PostgreSQL, IBM Db2,
- NoSQL databases - e.g. MongoDB, DyamoDB, Elasticsearch,
- Data Lake - e.g. S3,
- Analystic Application - e.g. Google Analytics, Microsoft Dynamics NAV,
- Collaboration Application - Slack, Microsoft Teams, Jira, Trello, GitHub,
- and many others.

The connectors are divided into two main groups: source and destination connectors. The purpose of the source connectors is to access data, download them and transform them to the JSON compatible with required by Airbyte. The role of source connectors is also to analyse the schema of the source data and enable choosing a part of them which should be transferred. For example, after discovering a schema, the user can choose to migrate only desirable tables from the database. The schema is also sent to the destination connector to ease data processing. Whereas the role of destination connectors is to save data in a destination. Firstly, the destination connector receives the schema created by a source connector. Based on the schema, the connector can perform required initialization operations. In the next step, the connector is receiving the data produced by the source connector as JSON files and transforms them into the data acceptable by the destination. Ultimately, the data is saved in the destination.

After configuring connectors, a user can establish a connection between a source and a destination connector for transferring data. In addition to choosing particular data from a source, the user can also define the mode of the saving data - overwrite or append (if both options are supported by the destination connector). Moreover, the user has the possibility to schedule periodic updating of the data in the destination.

Airbyte architecture

Figure 1.1 shows the main components in Airbyte architecture:

- `UI` - a graphical interface for interacting with the Airbyte API (`airbyte/webapp` container),
- `WebApp Server` - a component handling request from `UI` using `Config API` (`airbyte/server` container),
- `Config API` - an API handling operations like creating connectors and connections as well as managing configurations,
- `Config Store` - a component storing the information about connections (`airbyte/db` container),

- `Scheduler` - a component sending request to `Temporal service` to trigger synchronization; time intervals of synchronization are based on the configuration received from `Config API`; additionally it tracks success/failure of data synchronization (`airbyte/scheduler` container),
- `Scheduler Store` - a component storing statuses and job information for the `Scheduler` (`airbyte/db` container),
- `Temporal Service` - a component managing the task queue and workflows for the `Scheduler` (`airbyte/temporal` container),
- `Worker` - a component reading data from a `STDOUT` of source connector and writing it to a `STDIN` of destination connector (`airbyte/worker` container),
- `Temporary Storage` - a storage that workers can use to save temporary data on a disk.

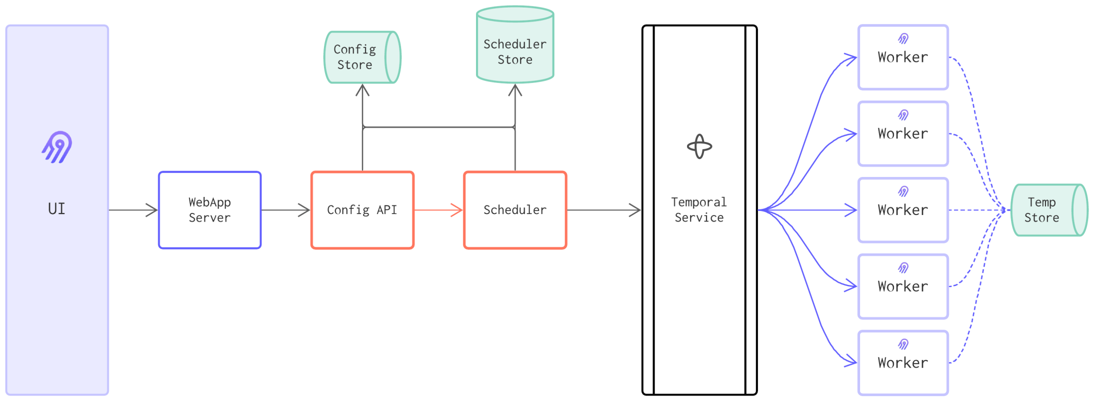

FIGURE 1.1: Airbyte architecture schema. Source: Airbyte documentation [2]

Data flow

Figure 1.2 shows the data flow during transferring data from source to destination. The main component responsible for executing jobs is `Worker` [3]. Initially, the `Worker` after receiving the signal from `Temporal Service` creates Docker containers with source and destination connectors. Then source connector connects to a source (e.g. MongoDB) and downloads data. Downloaded data is then transformed into JSON compatible with the earlier received schema and sent to its `STDOUT`. The data from the `STDOUT` is read by the `Worker` which afterwards writes them into the `STDIN` of the destination connector. The destination connector transform received JSONs into a format acceptable by the destination and writes the data. After source and destination connectors finish their work, the `Worker` shuts down containers with their corresponding connectors.

1.3.2 Apache Arrow

Overview

Apache Arrow is a cross-language library for in-memory analytics [4]. The crucial element of Apache Arrow is its in-memory columnar format. The columnar format is widely used while working with Big Data thanks to the possibility of doing calculations with high performance. Additionally, the Apache Arrow standardization allows developers to use the common data format

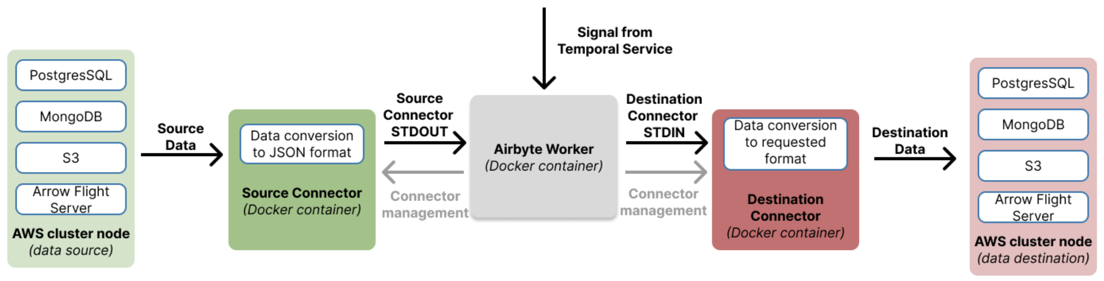
PostgresSQL
MongoDB
S3
Arrow Flight Server
AWS cluster node (data source)
Source Data
Data conversion to JSON format
Source Connector (Docker container)
Source Connector STDOUT
Connector management
Signal from Temporal Service
Airbyte Worker (Docker container)
Destination Connector STDIN
Connector management
Data conversion to requested format
Destination Connector (Docker container)
Destination Data
PostgresSQL
MongoDB
S3
Arrow Flight Server
AWS cluster node (data destination)

FIGURE 1.2: Airbyte dataflow schema

across different platforms and libraries. Thanks to that, the time required for serialization and deserialization between platforms was reduced.

Apache Arrow Flight

Apache Arrow Flight is a part of Apache Arrow dedicated to creating client-server architecture to transfer big amounts of data [6]. The Apache Arrow Flight use classes and objects from the standard library, which simplifies communication because no additional transformations are required. Data sent from client to server can be immediately used for further processing or be saved as files. The additional advantage of Apache Arrow Flight is the possibility of parallel transfers, which allows simultaneously streaming data to or from many servers.

Downloading data via Apache Arrow Flight

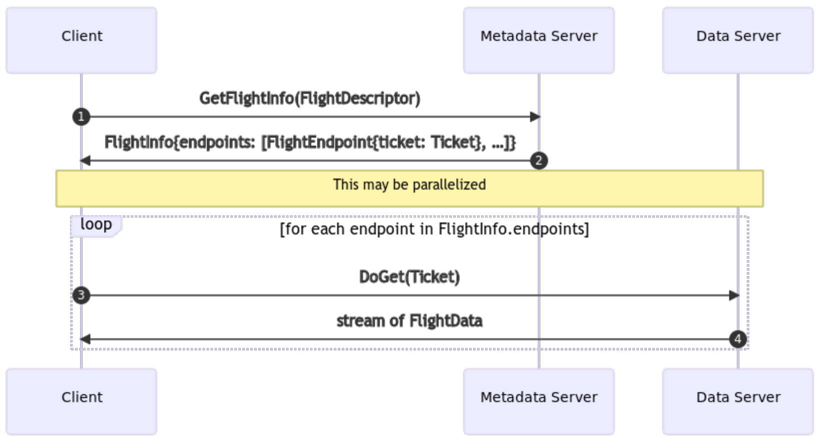
Client
Metadata Server
Data Server
GetFlightInfo(FlightDescriptor)
FlightInfo{endpoints: [FlightEndpoint{ticket: Ticket}, …]}
This may be parallelized
loop
[for each endpoint in FlightInfo.endpoints]
DoGet(Ticket)
stream of FlightData
Client
Metadata Server
Data Server

FIGURE 1.3: Apache Arrow Flight data downloading. Source: Apache Arrow Documentation [5]

Figure 1.3 shows the data flow for downloading data from the Apache Arrow Flight server [5]. Firstly, a client sends a request for metadata - addresses of the server where data is located. It is caused by the fact that the metadata can be stored on a different server than the proper data. Then the client can parallelly access data from servers (if data is available on different servers). After receiving a request from the client, the server resends a stream of Arrow record batches, which can be afterwards processed by the client.

Uploading data via Apache Arrow Flight

Figure 1.4 shows the data flow of uploading data to the Apache Arrow Flight server. The client sends all data as a stream of Arrow records, and the server can respond with custom metadata. Additionally, the first client request to the server also contains metadata about the incoming data.

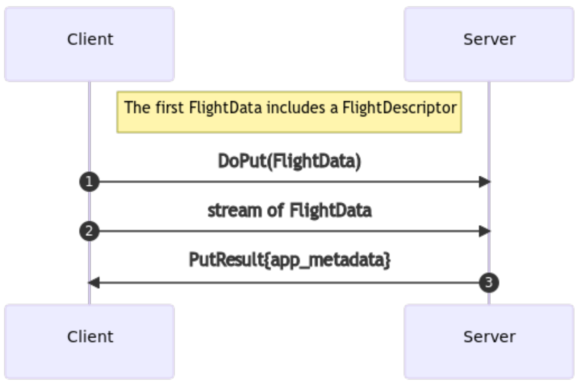

FIGURE 1.4: Apache Arrow Flight data uploading. Source: Apache Arrow Documentation [5]

Chapter 2

Implementation

The following chapter describes the implementation of the connector. Both connectors have been implemented in Python 3.9, which is the default version of Python supported by Airbyte. To ensure a proper possibility of the comparison between connectors, both had to be implemented in the same language.

The codebase can be found on the project's repository.

2.1 Source Connector

The role of the source connector is to transfer data from the Apache Arrow Flight server to Airbyte. However, before it is able to do that, it must check whether the parameters provided by a user allow it to connect to the source and discover the schema of the data. The structure of the main classes of the source connector is following:

2.1.1 SourceApacheArrow

SourceApacheArrow connector is an implementation of the `Source` class from Airbyte.

It implements the following methods:

- `check` - tests if the connector can establish a connection with the underlying data source with the user-provided configuration. The declaration of required credentials and parameters to run the connector is located in `spec.json`. The method returns a JSON object that reports whether using given by a user the credentials the connector is able to connect to the source, in this case, the Apache Arrow Flight server.

- `discover` - declares the different streams of data that the connector can produce. It returns a JSON object called an `AirbyteCatalog` that describes the data available in the source and their metadata like a list of fields and their types. Additionally, during the stream creation, the connector defines also available options of data replication - `Incremental` or `Full Refresh`.

- `read` - call the object of class `Client` that reads data from the Apache Arrow Flight server. The result of the method is a generator emitting the data from a source in a form of `AirbyteMessage` objects. It also filters out not selected by the user fields.

- `selected_fields` - it provides projections of selected data from a source.

2.1.2 Client

`Client` is a class responsible for communication with the Apache Arrow Flight server. The main aim of the `Client` class is to use the Apache Arrow library to reads data from the URL passed as a parameter by a user and convert them to a Python dictionary. The class also create `AirbyteStream` objects using a schema of the stored data.

2.2 Destination Connector

The role of the destination connector is to transfer data from Airbyte to the Apache Arrow Flight server. However, before it is able to do that, it must check whether the parameters provided by a user allow it to connect to the destination.

The structure of the main classes of the source connector is following:

2.2.1 DestinationApacheArrow

Destination connector is an implementation of the `Destination` class from Airbyte.

It implements the following methods:

- `check` - Its goal is to validate configuration parameters passed by the user;
- `write` - It creates and performs flows of data using the following classes enabling saving the data read by the source connector to the Apache Arrow Flight server.

2.2.2 DestinationApacheArrowConfig

The `DestinationApacheArrowConfig` class ensures that all parameters required for the destination are present and correct. The declaration of required credentials and parameters to run the connector is located in `spec.json` as with the source connector. The role of the class is also to establish a connection with the Apache Arrow Flight server based on the given parameters.

The parameters required by the connector are:

- `destination_server` - server's URL,
- `destination_path` - path to the data on the server,
- `chunk_size` - chunk size used in the data transfer;

2.2.3 DestinationApacheArrowRecordConsumer

The `DestinationApacheArrowRecordConsumer` class is responsible for the creation of `File Writers` for each stream. Afterwards, it accepts an iterable object of `AirbyteMessage` which is passed to the proper `File Writer` based on `stream` the stream attribute of the message.

When all the messages have been accepted, it closes the connection with an Apache Arrow Flight server.

2.2.4 DestinationApacheArrowFileWriter

`DestinationApacheArrowFileWriter` is the main class of the destination connector.

The three main tasks of the class are:

- creating the target schema of the data based on data provided by the source connector,
- converting data received from Airbyte to the format enabling sending them to an Apache Arrow Flight server,
- sending data to an Apache Arrow Flight server in batches with given size.

Each `AirbyteMessage` is transformed with dictionary comprehension to the dictionary of values (to ensure high performance) and added to the end of the batch list of `DestinationApacheArrowFileWriter` object.

If the file writer batch list length is equal to the given batch size (or if all messages from the source were received), the list of dictionaries is transformed into Arrow format using `RecordBatch.from_pylist` method and sent to an Apache Arrow Flight server. This way of processing the received data enables to perform operations efficiently.

Chapter 3

Performance analysis

3.1 Test environment

The tests were performed on a virtual machine, which was created using Amazon Elastic Compute Cloud (EC2) available on Amazon Web Services (AWS). AWS is the world's most comprehensive and broadly adopted cloud platform thanks to which tests are executed in a stable and reproducible environment.

The virtual machine was using Ubuntu Server 22.04 LTS image as the operating system. The EC2 instance had the following resources

- `c5.2xlarge`
- `8 vCPUs`
- `16 GiB Memory`
- `Storage - 64 GiB`

3.2 Test data

The single record of exemplary data consisted of the following fields:

- `id` - Integer
- `first_name` - String
- `last_name` - String
- `age` - Integer
- `phone_numbers` - List of Strings
- `income` - Double
- `house_location` - object containing two fields:
  - `longitude` - Double
  - `latitude` - Double

The size of the one record approximately was equals to 128B (based on the size of CSV files stored in S3 bucket)

3.3 Testing process

The tests investigated the communication of Airbyte connectors with 4 different systems to store data:

- Apache Arrow Flight - the main object of analysis,
- PostgreSQL - an example of a SQL database,
- MongoDB - an example of a NoSQL database,
- S3 - example of cloud storage.

Table 3.1 presents programming languages used to develop connectors to the mentioned systems. Nearly all of them were created using Python or Java.

TABLE 3.1: Programming language of Airbyte connectors

Platform	**Source connector language**	**Destination connector language**
Apache Arrow Flight	Python	Python
PostgreSQL	Java	Java
MongoDB	Ruby	Java
S3	Python	Java

In the EC2 instance, local servers were created for MongoDB, Apache Arrow Flight and PostgreSQL. To store data in S3 data, the bucket was created using AWS Console. Before the tests, all data sources were filled with example data with various data types described in the 3.2 section. In the case of S3, the data was stored as CSV files.

The first type of tests measured the time in which data will be retrieved from the source, processed by the source and destination connector and saved in the indicated place. This process was subjected to various sizes of data - starting at 50.000 records up to 1.600.000 records. For each data size, the tests were performed several times, and then the mean result and standard deviation were calculated. In this test 12 different source and destination connectors combination were tested:

- Comparison of connections consisting of the same connectors
  - `Apache Arrow Flight - Apache Arrow Flight`
  - `PostgreSQL - PostgreSQL`
  - `MongoDB - MongoDB`
  - `S3 - Apache Arrow Flight`
- Comparison of connections consisting of different source connectors and Apache Arrow Flight destination connector
  - `Apache Arrow Flight - Apache Arrow Flight`
  - `PostgreSQL - Apache Arrow Flight`
  - `MongoDB - Apache Arrow Flight`
  - `S3 - Apache Arrow Flight`
- Comparison of connections consisting of different destination connectors and Apache Arrow Flight source connector

– `Apache Arrow Flight - Apache Arrow Flight`
– `Apache Arrow Flight - PostgreSQL`
– `Apache Arrow Flight - MongoDB`
– `Apache Arrow Flight - S3`

The second type of tests measured the impact of different batch sizes selected in the Apache Arrow Flight destination connector on the total time of processing. These tests were executed on the same size (800.000 records) and the same type of data (connection from Apache Arrow Flight to Apache Arrow Flight).

3.4 Results of measurements

3.4.1 Comparison of connections consisting of the same connectors

Figure 3.1 shows the graph presenting the result of tests comparing the time needed to transfer data from a data source to the same platform using Airbyte. For example, the Airbyte connection was configured to read data from MongoDB and save them also to MongoDB. The aim of these 4 tests was to analyse the cost of operation executed by Airbyte during moving data between the same type of data stores.

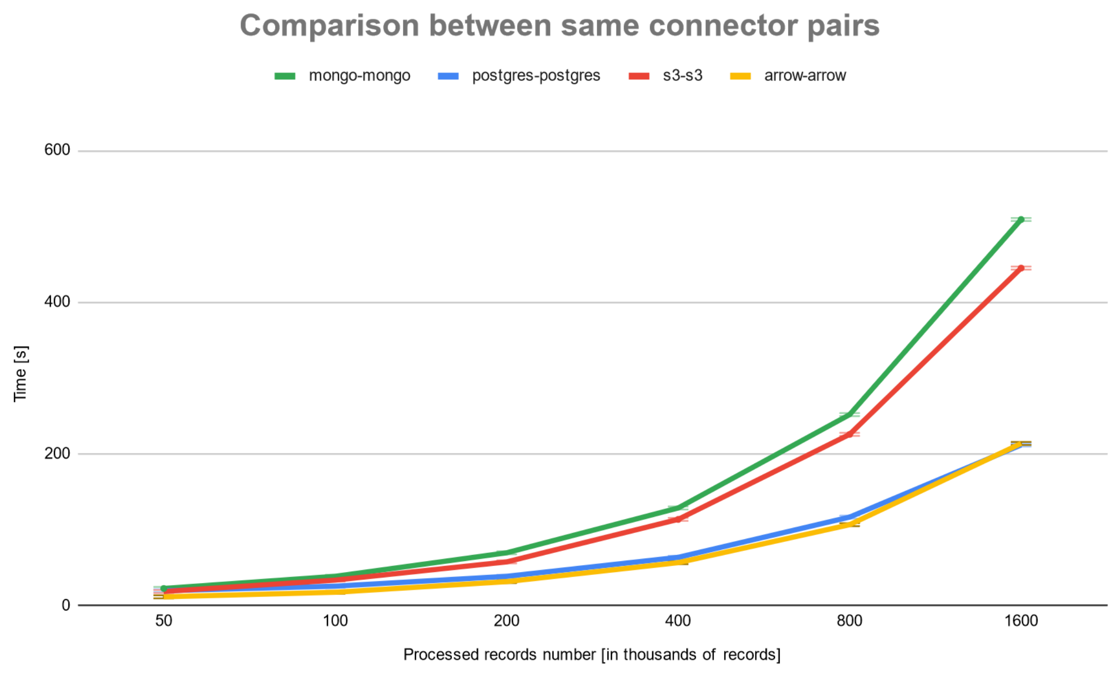

FIGURE 3.1: Comparison between the same types of connectors

The results show that two types of connections, from MongoDB to MongoDb and from S3 to S3, achieved worse results than the other two connections. For the number of records equal to 1.600.000, the results for them are over two times longer than the results achieved by the connection from PostreSQL to PostreSQL and connection from Apache Arrow Flight to Apache Arrow Flight. The further analyses done in points 3.4.2 and 3.4.3 can be an explanation for the worse results for these two connections. In the case of the connection from S3 to S3 the reason is the source connector and in the case of the connection from MongoDB to MongoDB, the reason is the destination connector.

3.4.2 Comparison of connection consisting of different source connectors and Apache Arrow Flight destination connector

Figure 3.2 shows the graph presenting the result of the tests comparing connections consisting of different source connectors and the created Apache Arrow Flight destination connector. The aim of these 4 tests was to compare the created Apache Arrow Flight source connector to other source connectors already implemented in Airbyte.

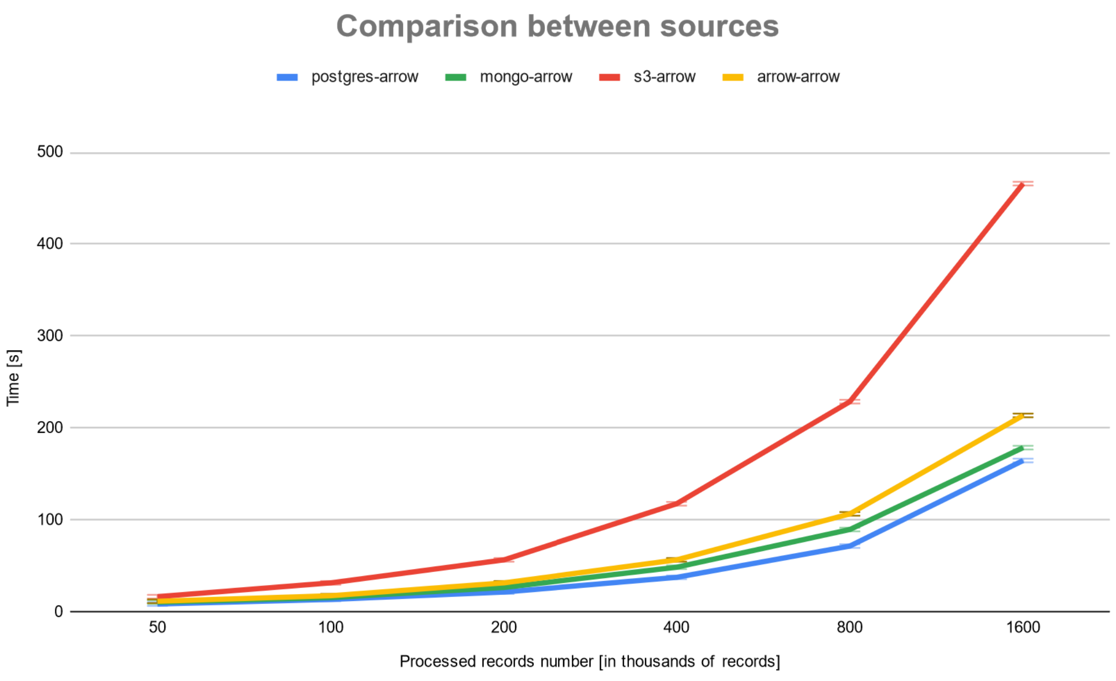

FIGURE 3.2: Comparison between different source connectors and Apache Arrow Flight destination connector

The test result shows that connections using the Apache Arrow Flight source connector achieve results comparable to connections using PostgreSQL and MongoDB source connectors. The worst result achieves the connection using the S3 source connector, which is over two times worse than the connection with the Apache Arrow Flight source connector.

The poor performance of the S3 source connector may be caused by the facts that:

- S3 data was gathered from Amazon S3 Bucket and not from localhost like in the case of other connections,
- S3 source connectors parse each line of CSV files using PyArrow which firstly create data column-oriented format which is then converted to the row-oriented format.

3.4.3 Comparison of connections consisting of different destination connectors and Apache Arrow Flight source connector

Figure 3.3 shows the graph presenting the result of the tests comparing connections consisting of the created Apache Arrow Flight source connector and different destination connectors. The aim of these 4 tests was to compare the created Apache Arrow Flight destination connector to other destination connectors already implemented in Airbyte.

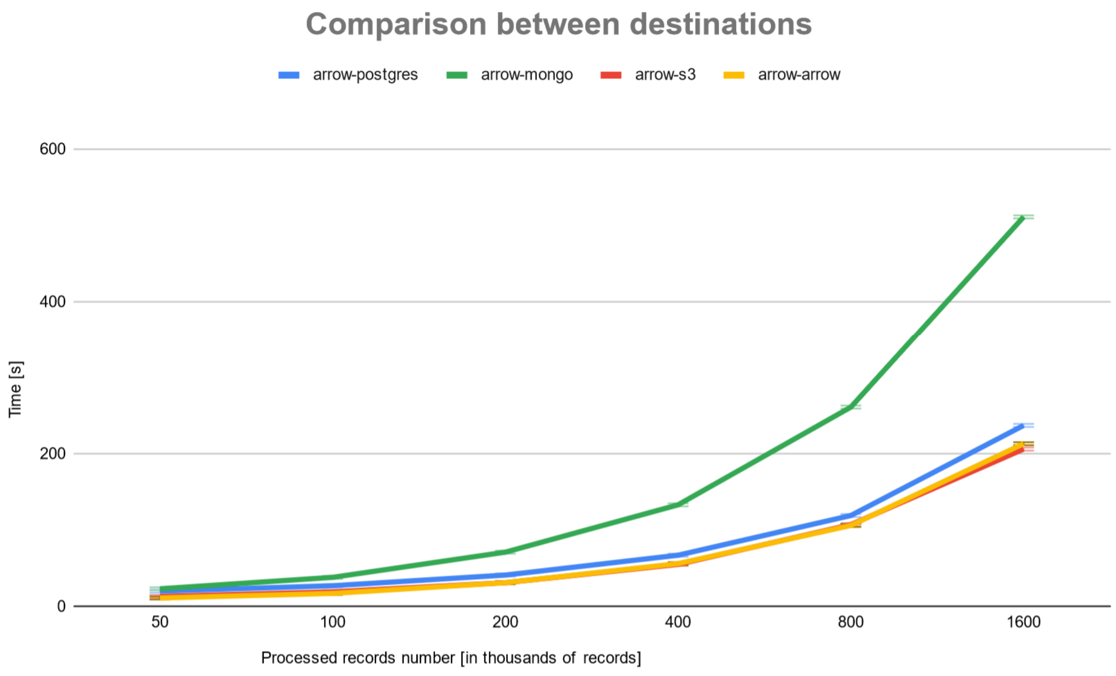

FIGURE 3.3: Comparison between Apache Arrow Flight source connector and different destination connector

The test result shows that connections using the Apache Arrow Flight destination connector achieve results comparable to connections using PostgreSQL and S3 destination connectors. The worst result achieves the connection using the MongoDB destination connector, which is over two and half times worse than the connection with Apache Arrow Flight source connector.

The poor performance of MongoDB destination connector may be caused by its implementation. Initially, the connector creates from each received data record a document which is inserted into a temporary collection. Afterwards, it reads the whole temporary collection and finally inserts all documents at once into the proper collection.

3.4.4 Impact of different batch size

Figures 3.4 and 3.5 show the graphs presenting the results of tests using the different batch sizes in the Apache Arrow Flight destination connector. The tests were performed for 800.000 records and a connection from Apache Arrow Flight to Apache Arrow Flight.

Figure 3.4 describes the time of the batch creation. In any case, the batch creation time wan not longer than one second. However, it is visible that the batch processing time decreased with the increase of batch size, but only to some certain value. After this value was reached, the time of batch processing increased. That behaviour may be caused by the size of the CPU cache.

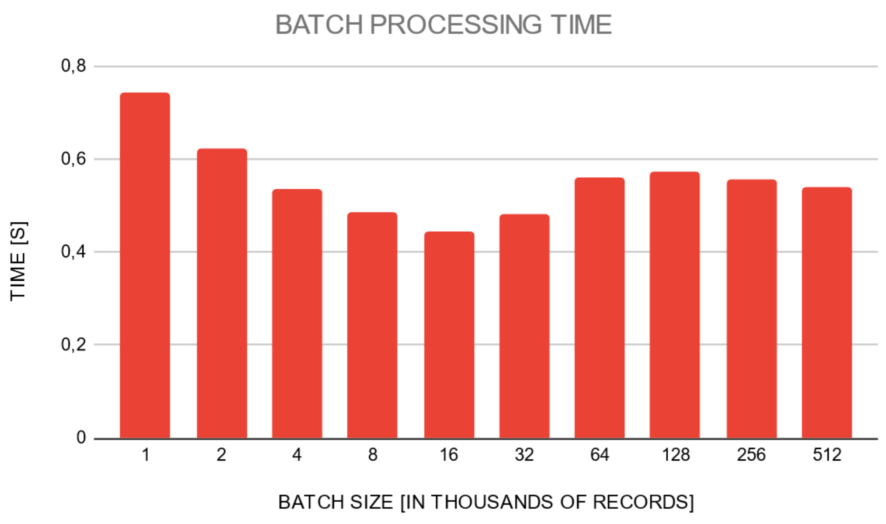

FIGURE 3.4: Batch processing time

Nevertheless, figure 3.5 presents that the size of the batch has very little effect on the total processing time. The time of data processing is nearly the same for each batch size. It is caused by the fact that for processing 800.000 records, the time of batch processing is equal to only around 0.5 percent of the total time spent on whole processing.

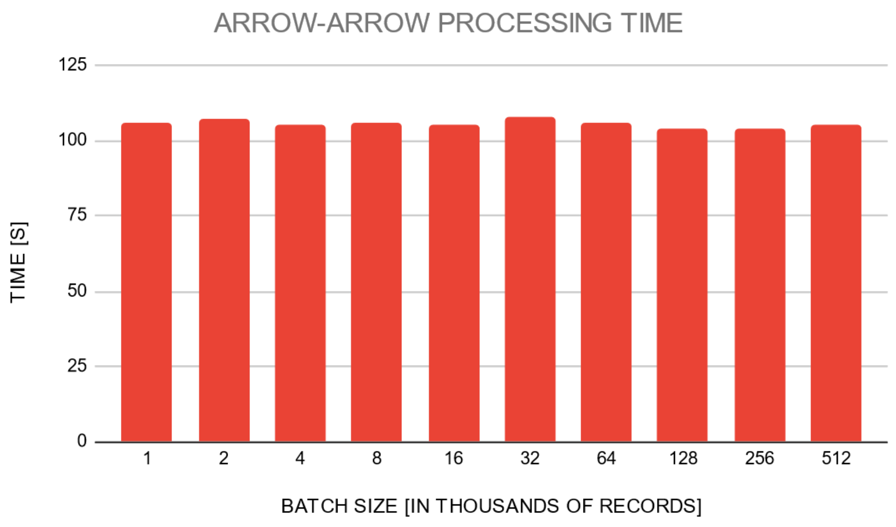

FIGURE 3.5: Total processing time for different batch size

Chapter 4

Conclusions

The implementation process, performance tests and results analysis enabled us to reach a few conclusions about Airbyte and Apache Arrow Flight.

Firstly, both technologies are relatively new and still intensively developing. It concerns especially Airbyte, whose documentation does not always describe the current state of the project, but also shows the feature not implemented yet or the in development phase.

One of the biggest disadvantages of Airbyte during development was the lack of complex data types. In the documentation there was information on data types like, for example, Date, Datetime with timezone, and Big Integer, however, during the development of Apache Arrow Flight connectors, Airbyte implemented only 5 of them - String, Boolean, Number, Array, and Object. The mentioned data types were introduced to the Airbyte code only in June 2022, so after the performing tests. The newly introduced data types are represented as a string with additional metadata in compliance with earlier written documentation. For this reason, there nearly all Airbyte connectors do not support, for example, detecting dates.

That show an additional aspect of Airbyte connectors - all the connectors are still in the development phase. For example, MongoDB and Postgres connectors are in alpha, and S3 connectors are in beta/generally available version. Because of that fact, the usage of some connectors comes with the risk of instability and may cause a need to update them multiple times during the development process like in the case of this project.

What is more, Airbyte and Apache Arrow have a problem with the inconsistency in API between programming languages, which makes developing connectors in different languages more difficult (e.g. `ArrowRecordBatch` in Java and `RecordBatch` in Python).

Additionally, the Java implementation of Apache Arrow 7.0 uses operations using a reflection mechanism during saving the data to a file. The mechanism is limited with newer Java versions and causes problems during attempts of creating Airbyte connectors using required by Airbyte Java 17 because of the lack of possibility to open the required module.

Moreover, the performance tests showed that the results achieved by Apache Arrow Flight source and destination connector are comparable to the results of different connectors. The test result presented also that the poor implementation of some connectors caused the discrepancy in performance when compared to the rest of the tested connectors. If one of the connectors is significantly slower than the second one, its poor performance negatively impacts the whole process of data transfer using that connection.

Furthermore, the test showed that batch conversion does not have a significant impact on the total time of processing a big amount of data

To sum up, Apache Arrow Flight connectors have the potential to become useful in the already created Apache Arrow data flow, or for the newly developed flows. Airbyte allows developers to connect Apache Arrow Flight servers with many various data sources. Efficiency reached by created connectors is comparable to other already introduced connectors or even in some cases better. In addition, moving processing to Apache Arrow Flight servers allows developers for quick, in-memory columnar data processing to reach needed results more efficiently.

Bibliography

[1] Airbyte. Airbyte Documentation, 2022. `https://docs.airbyte.com/` (Accessed June 09, 2022).

[2] Airbyte. Airbyte Documentation: Architecture overview, 2022. `https://docs.airbyte.com/understanding-airbyte/high-level-view/` (Accessed June 09, 2022).

[3] Airbyte. Airbyte Documentation: Workers & Jobs, 2022. `https://docs.airbyte.com/understanding-airbyte/jobs` (Accessed June 09, 2022).

[4] Apache Software Foundation. Apache Arrow Documentation, 2022. `https://arrow.apache.org/docs/index.html` (Accessed June 09, 2022).

[5] Apache Software Foundation. Apache Arrow Documentation: Arrow Flight RPC, 2022. `https://arrow.apache.org/docs/format/Flight.html` (Accessed June 10, 2022).

[6] Wes McKinney. Introducing Apache Arrow Flight: A Framework for Fast Data Transport, 2019. `https://arrow.apache.org/blog/2019/10/13/introducing-arrow-flight/` (Accessed June 09, 2022).