Second Assignment - Working on Data Ingestion Features

Last modified: 16.10.2019

By Linh Truong(linh.truong@aalto.fi)

1 Introduction

The goal of this assignment is to develop data ingestion features in big data platforms.

2 Constraints and inputs for the assignment

In this assignment, we assume that you operate a big data platform **mysimbdp**, of which **mysimbpd-coredms** is the component for big data store (databases and storage). You might also have **mysimbdp-daas**, which provides REST API for **mysimbdp-coredms**, already implemented. Overall, you have a set of APIs for your customers to store data into **mysimbdp** and the data will be stored in **mysimbdp-coredms** as the final sink.

> Note: **mysimbdp** is the name of your big data platform. It has many components.

Furthermore, we assume that you have different customers who need to ingest their data into **mysimbdp**. Each customer has its own data (different syntaxes and/or semantics) but we assume that all your customers use the same database/storage model, e.g., document-based, column family -based or file -based models. Your **mysimbdp** supports multi-tenancy, thus you can have one deployment of **mysimbdp** for many customers.

With the above-assumption, your customers can design how they would store the data to **mysimbdp** using the APIs and components you offer to them. Some components will be implemented in this assignment.

> **Important note**: in this view, you can reuse your previous implementation of **mysimbdp-coredms** and **mysimbdp-daas**. However, we consider them as external, reusable components. To simplify the test and development, you can have their code within this assignment but you need to arrange their code in clear, separate modules, as they cannot be taken into account for grading. This assignment, however, does not reply on the first assignment. You can select suitable technology for your **mysimbdp-coredms**, which offers APIs for storing data.

We will continue with the set of technologies and programming languages like in the first assignment. You can continue to use previous datasets for testing:

> Note: Keep in mind that you have different customers. So that you must not allow all customers using the same dataset in your tests. One of your customer will use the dataset listed in the first assignment. Other customers will use different datasets, either collected from Internet or created from the existing dataset.

In this assignment, all customers of your platform will ingest their data into **mysimbdp** (the final data sink is **mysimbdp-coredms**). They can ingest using batch ingestion and near-realtime ingestion features. Near-realtime ingestion must be done via message brokers. You can use one of the following frameworks/technologies for message brokers in **mysimbdp**:

- AMQP with RabbitMQ
- MQTT with Mosquitto
- Apache Kafka

In **mysimbdp** each customer will have data specified in only one data model/schema. All types of ingestion (batch and near-realtime) will insert data into the data store with the same data model. It means data can be stored in different databases/tables or files but all data of the same customer in **mysimbdp** follow the same data model.

To simplify the ingestion implementation, we also assume that an ingestion of data is stateless, isolated and immutable. It is important to note that customer data sources and platforms are distributed in different administrative domains.

3 Requirements and delivery

> Important note: **implementation** should not be interpreted that the implementation starts from scratch. Reuse is important. But you should check the requirements carefully, because often you cannot reuse existing tools by just **configuring** them. Usually you need to implement extras to work with existing tools.

Part 1 - Ingestion with batch (weighted factor for grades = 2)

1. The ingestion will be applied to files of data. Define a set of constraints for files which should (or should not) be ingested and for the customer service profile w.r.t. ingestion constraints (e.g., number of files and data sizes). Implement these constraints into simple configuration files (e.g., JSON or YAML) (1 point)

2. Each customer will put files to be ingested into a directory, **client-input-directory**. Implement a component **mysimbdp-fetchdata** that automatically detect which files should

be ingested based on customer profiles and *only move* the files from **client-input-directory** inside **mysimbdp** (only staging data in; data has not been stored into **mysimbdp-coredms** yet). (1 point)

3. Each customer provides a simple program, **clientbatchingestapp**, which will take the customer's files as input and ingest the files into the final sink **mysimbdp-coredms**. Design and develop a component **mysimbdp-batchingestmanager** that invokes customer's **clientbatchingestapp** to perform the ingestion when files are moved into **mysimbdp**. **mysimbdp** imposes the model that **clientbatchingestapp** (and the customer) has to follow. (1 point)

4. Develop test programs (**clientbatchingestapp**), test data, and test profiles for customers. Show the performance of ingestion tests, including failures and exceptions, for at least 2 different customers in your test environment. (1 point)

5. Implement and provide logging features for capturing successful/failed ingestion as well as metrics about ingestion time, data size, etc., for files which have been ingested into **mysimbdp**. The log should be outputed in a separate file or database for analytics of ingestion. (1 point)

Part 2 - Near-realtime ingestion (weighted factor for grades = 2)

1. For near-realtime ingestion, you introduce a simple message structure **ingestmessagestructure** that all consumers have to use. Design and implement **ingestmessagestructure** for your customers; explain your design. (1 point)

2. Customers will put their data into messages following **ingestmessagestructure** to send the data to a message broker, **mysimbdp-databroker** (provisioned by **mysimbdp**) and customers will write a program, **clientstreamingestapp**, which read data from the broker and ingest data into **mysimbdp**. Provide a component **mysimbdp-streamingestmanager**, which invokes on-demand **clientstreamingestapp** (e.g., start, stop). **mysimbdp** imposes the model that **clientstreamingestapp** has to follow. (1 point)

3. Develop test programs (**clientstreamingestapp**), test data, and test profiles for customers. Show the performance of ingestion tests, including failures and exceptions, for at least 2 different customers in your test environment. (1 point)

4. **clientstreamingestapp** decides to report the its processing rate, including average ingestion time, total ingestion data size, and number of messages to **mysimbdp-streamingestmanager** within a pre-defined period of time. Design the report format and the communication mechanism for reporting. (1 point)

5. Implement the feature to receive the report from **clientstreamingestapp**. Based on the

report from **clientstreamingestapp**, when the performance is below a threshold, e.g., average ingestion time is too low, or too many messages have to be processed, **mysimbdp-streamingestmanager** decides to create more instances of **clientstreamingestapp**. Define a model for specifying constraints for creating/removing instances for each customer. (1 point).

Part 3 - Integration and Extension (weighted factor for grades = 1)

> Notes: no software implementation is required for this part

1. Produce an integrated architecture for both batch and near-realtime ingestion features in this assignment and explain the architecture. (1 point)
2. Assume that if a file for batch ingestion is too big to ingest using the batch ingestion implementation in Part 1, design your solution to solve this problem by reusing your batch or near-realtime components in Parts 1 or 2. (1 point)
3. Regardless of programming languages used, explain why you as a platform provider should not see the code of the customer, like **clientbatchingestapp** and **clientstreamingestapp**. In which use cases, you can assume that you know the code? (1 point)
4. If you want to detect the quality of data and allow ingestion only for data with a pre-defined quality of data condition, how you, as a platform provider, and your customers can work together? (1 point)
5. If a consumer has multiple **clientbatchingestapp** and **clientstreamingestapp**, each is suitable for a type of messages or files, how would you extend your design and implementation in Parts 1 & 2 (only explain the concept/design) to support this requirement. (1 point)

Bonus points

> Note: the bonus point has to be documented clearly for grading

- If you can design and implement the dynamic management of instances for Part 2/Question 5, you get 5 points
- If you can develop the solution that automatically switch from batch to microbatching based on a set of constraints (e.g., file size, time, bandwidth) you get 5 points

4 Other notes

Remember that we need to **reproduce** your work. Thus:

- Remember to include the (adapted) deployment scripts/code you used for your installation/deployment
- Explain steps that one can follow in doing the deployment (e.g. using which version of which databases)
- Include logs to show successful or failed tests/deployments
- Include git logs to show that you have incrementally solved questions in the assignment
- etc.

Appendix

The following information is extracted from the first assignment.

Technologies for data store:

- [MongoDB] (https://www.mongodb.com/)
- [ElasticSearch] (https://www.elastic.co/)
- [Hadoop File System] (https://hadoop.apache.org/)
- [Cassandra] (http://cassandra.apache.org/)

Datasets as input data

- The list of datasets: (https://version.aalto.fi/gitlab/bigdataplatforms/cs-e4640-2019/tree/master/data)

Programming languages:

- Python
- Scala
- JavaScript/NodeJS
- Java
- GoLang