Software Architecture Project 2 Report

KAFKA

Pedro Marques, 92926 Inês Leite, 92928

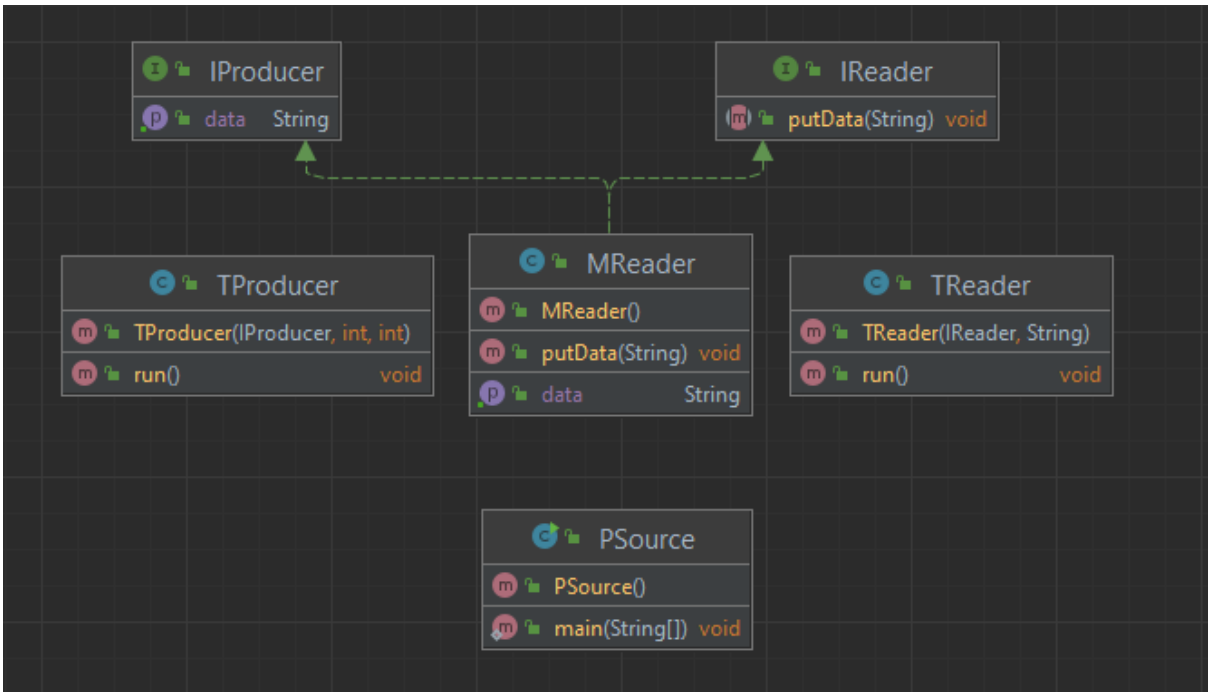
Introduction

This report provides an overview of the project that we developed, related to the second practical assignment about the Apache Kafka platform. For each use case defined in the work description, we created a UCn package, using n as the number of the use case. Inside the source folder, there's a main package called UCn, again with n as the user case number which contains 3 inner packages: PSOURCE, PPRODUCER and PCONSUMER, along with a AppConstants class with public values, because they contain global variables like the java socket for the servers, number of kafka producers, number of consumers and number of consumer groups.

Diagrams

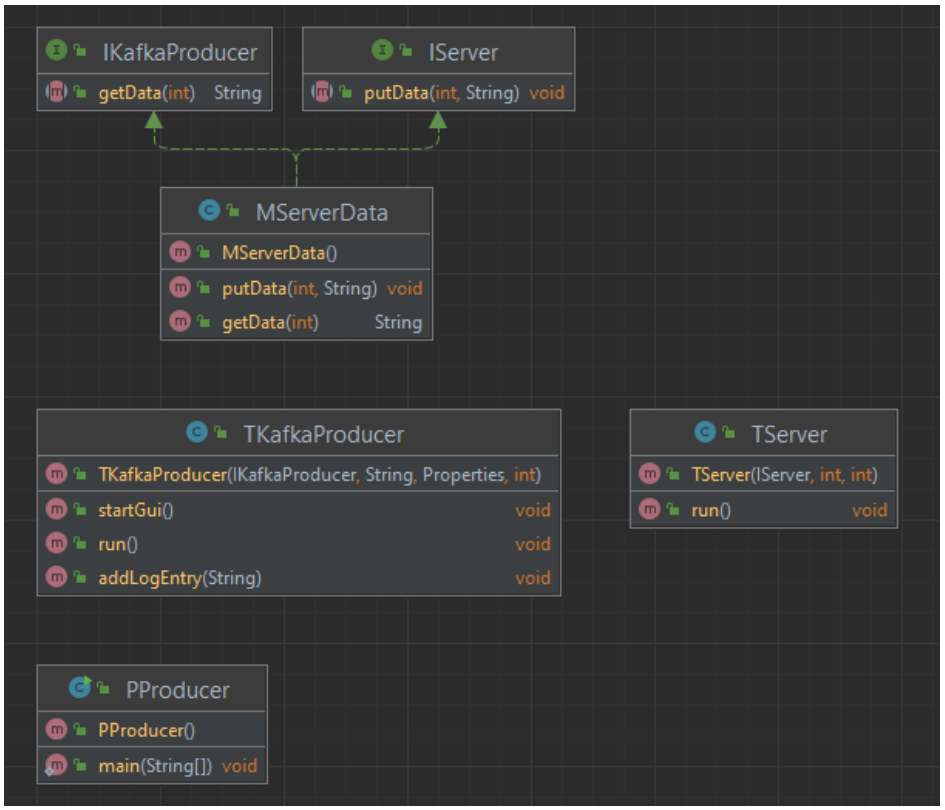
Seeing that most use cases are similar with slight differences, we created the following base class diagrams for each process.

PSOURCE Process



For this process, the PSource class instantiates every other object in order to achieve its goal of reading from the Sensor.txt file and sending the data to the Kafka Producer. For this we use a monitor which we called MReader. A single TReader thread reads every 3 lines from the file and concatenates them into a single record, which then proceeds to put in the data queue of the MReader. Next, the idle producers who were waiting for new data are signalled and proceed to send this new data to the PPRODUCER process via java sockets. Each producer sends data to a specific server with the same id, which is simply an incremental integer.

PRODUCER Process

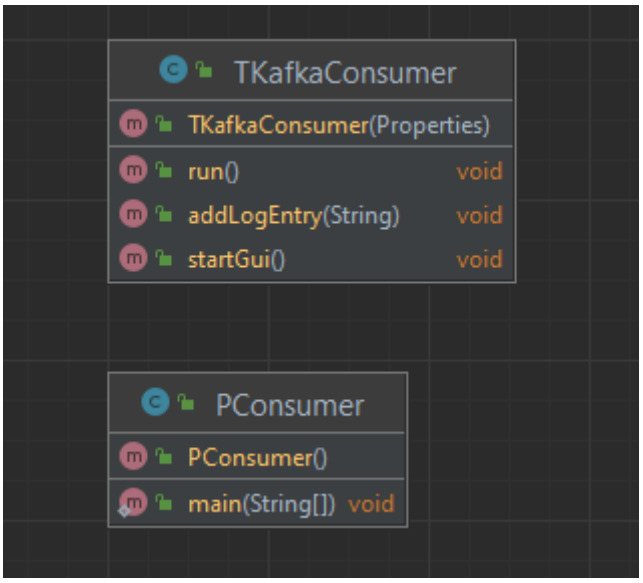


In this case, the PProducer class is responsible for instantiating every other object in order to receive data from the PSource process and sending it to the kafka cluster. In this class we also define the properties of each Kafka Producer. In order to achieve the UC goals.

A monitor, MServerData is used to share the data received between the server sockets and the TKafkaProducer threads. Each server socket receives data from each producer from the PSource process and also sends it to a single TKafkaProducer. All these objects contain the same id (incremental integer), which allows for a single channel of communication between PSource.TProducer, PProducer.TServer and PProducer.TKafkaProducer. The monitor contains a data queue for each TKafkaProducer, as such, each server puts the data in the shared region in the queue corresponding to its assigned TKafkaProducer. These last threads, which were previously idle waiting for data, retrieve it from the shared region and publish it in the Kafka Cluster.

Each TKafkaProducer contains a GUI which includes at least the number of records it received, a list with all the records and a counter of records received for each Sensor.

PCONSUMER Process



Finally we have the PConsumer process which intends to retrieve data from the Kafka cluster and display it. The PConsumer class is responsible for instantiating every TKafkaConsumer thread along with the Kafka Consumer properties. The TKafkaConsumer retrieves the data from the Kafka Cluster and displays it in its own GUI. Each interface contains at least the number of records it received, a list with all the records and a counter of records received for each Sensor.

Use Cases

Use Case 1

Goal: Records need to preserve ordering, records can be lost. Default performance
In order to preserve data order without using the idempotence property, the property max.in.flight.requests.per.connection with default value 5 is set to 1. This property defines the maximum number of unacknowledged requests the client will send on a single connection before blocking. Seen that the idempotence property is false by default, if the value of this property was bigger than 1 then there could be data reordering. In order to prevent an infinite loop of retries and risking data loss, we set the number of retries property at 100. Finally, we kept the default value of the delivery.timeout.ms property to achieve default performance.

Since the first property only ensures correct data ordering in clusters with 1 partition, every push made uses the same key, to further ensure that the order is the same because data published with the same key maintains order in the KafkaCluster.

Use Case 2

Goal: Minimise latency. Records can be lost, but minimise the possibility of losing all records of a sensor ID.

To minimise Latency, the following properties were used: linger.ms=0, compression.type=none, because compression isn't suitable for low latency applications,

and, lastly, acks=1 which means the leader broker responds sooner to the producer before all replicas have received the message.

To ensure the second goal, in the PProducers.java class the data sent to the consumer was directed to a specific partition (data from sensor1 went to partition 5, sensor 2 to partition 4, and so on).

Use Case 3

Goal: maximum throughput, records can be reordered, minimise the possibility of losing records but while minimising the impact on the overall performance. Records can be reprocessed but try to avoid some degree of reprocessing.

In order to achieve our goal we changed the values of the batch.size, linger.time.ms and compression.method properties of the Producer. These properties allowed us to send more data with each KafkaProducer publish, increasing throughput. We changed the batch size from the default 163834 to 200000, the linger time from 0ms to 100ms and the compression method from none to lz4, because lz4 has the best performance over all the methods. Furthermore, to minimise the possibility of losing records but while minimising the impact on the overall, we changed the acks property to 1 from the default all. As such, the leader will only have to wait for the publish confirmation of one of the followers. In this case, there is the possibility of records being lost if the leader were to fail before the followers replicated the record.

Furthermore, in order to maximise throughput, the consumer fetch.max.wait.ms and fetch.min.bytes properties were also increased.The first one defines a maximum threshold for time-based batching and has the default value of 500ms which we increased to 1000ms. The latter sets a minimum threshold for size-based batching and we increased it from the default 1 to 100000. Furthermore, records can be reprocessed but some degree of reprocessing should be avoided. As such, the auto.commit.interval.ms property defines the frequency that the consumer offsets are auto-committed to Kafka and is decreased from the default 5000ms to 1000ms. This, with the enable.auto.commit property set to true (which is by default) guarantees that a certain degree of reprocessing is avoided.

Use Case 4

Goal: Records cannot be reordered and minimise the possibility of losing records. For the first goal, as requested, we did not use the property of idempotence the max.in.flight.requests.per.connection property was set to 1 to speed up the process and the property of retries as set with a large number that signifies the number of retries when resending a failed message request. To minimise the possibility of losing records three properties were added:acks=all to force a partition leader to replicate messages to a certain number of followers before acknowledging that the message request was successfully received; and min.insync.replicas=2 because it sets the numbers of brokers that need to have logged a message before an acknowledgment is sent to the producer.

Use Case 5

Because this use case doesn’t define any desired performance or availability goal, all properties are left as default. Furthermore, it is intended to use the voting replication tactic to compute the maximum and minimum temperatures. This tactic intends on removing random

hardware failures, by executing the same function multiple times with the same input. In our case, the final result is the one which occurs the most, if there is no majority, the first one is the selected. As such, a new variable in AppConstants called noVotingReplicationTries defines how many times the same computation must be executed for this tactic. The GUI then presents the maximum and minimum temperature for each sensor in the same area where it presents the counter of records.

Use Case 6

As UC5, this use case doesn't define any desired performance or availability goals, so all properties are left as default. However, it's intended to compute the average temperature for each sensor, which is shown in the GUI.

What has not been implemented?

Unfortunately, we were not able to complete the UC2.

Contribution

Pedro Marques: 60%
Inês Leite: 40%

Bibliography

- https://docs.confluent.io/cloud/current/client-apps/optimizing/latency.html
- https://strimzi.io/blog/2021/01/07/consumer-tuning/
- https://strimzi.io/blog/2020/10/15/producer-tuning/
- https://lotusflare.com/news/managing-kafka-offsets-to-avoid-data-loss/