Software Architectures

Practical Assignment 2 Report

Leandro Silva, 93446
Margarida Martins, 93169

UC1

In the first use case there is only one producer and one consumer in one partition. The main constraint is that the consumer needs to receive the records in the original order.

In order to do this we setted:

- `retries` to 0. This means that the producer can automatically retry in case of a missed send. With this and because there is only one partition, it is guaranteed that the records will arrive in the same order.
- `acks` to 0. This means that the producer will not receive the confirmation that the message was read. This is because the use case allows for data to be lost.

UC2

In the second use case there are 6 producers and 6 consumers and 6 partitions replicated in three brokers. This means that if The main constraints are to minimize latency and assign for each consumer only one sensor ID data.

In order to do this we setted:

- `batch.size` to 1 and `linger.ms` to 1. Both these properties allow the producer to not wait to fill up while reducing the size to fill up to the minimum. This will decrease latency.
- 6 partitions each corresponding to a sensor ID. The producers upon receiving a record will send it in the partition corresponding to the sensor ID. Each consumer is assigned to only one partition fulfilling the constraint of receiving only one sensorID data.
- 3 brokers with 3 replicas. This means that if a partition is no longer available, it will still keep available with the messages not synced lost. This results in a very unlikely loss of all data for a sensor ID. The property `unclean.leader.election.enable` was set to true in order to allow for an out-of-sync replica to become the leader.
- `acks` set to 1, meaning that the producer will consider the write successful when only the leader receives the record. Since not all replicas are acknowledged, this can result in some data loss.
- `retries` to 0, since some data loss is tolerable.

UC3

In the third use case there are 3 producers and 3 consumers in the same consumer group. This use case aims to achieve high throughput while minimizing the loss of data and performance. There should not exist much reprocessing.

In order to do this we setted:

- `max.in.flight.requests.per.connection` to 10, with this the messages can be reordered but there will be no need to wait for acknowledgements until there are more than 10 unacknowledged.
- `acks` set to 1, a middle value to compromise loss of records with performance.
- `batch.size` set to 100000 bytes, since larger batch sizes result in fewer requests to Confluent Cloud, which reduces load on producers and the broker CPU overhead to process each request.
- `linger.ms` set to 10 ms. This delays the producer to give more time for batches to fill before sending. The trade-off is tolerating higher latency as messages aren't sent as soon as they are ready to send.

UC4

In the third use case there are 6 producers, 6 consumers, 6 partitions and 3 brokers. In this use case records can not be reordered and the loss of data should be minimized. Each producer will only send messages to one partition in which only one consumer is assigned, this schema allows for the data to arrive with the same order as the source file.

In order to do this we setted:

- `retries` to 0. This means that the producer can automatically retry in case of a missed send. It is guaranteed that the records will arrive in the same order.
- `acks` to all. This means that the producer will always receive the confirmation that the message was read. With this data won't be lost.
- 3 replicas at least 2 of each in-sync. This will prevent data loss in case of a broker failure.

UC5

Since we were constrained to 6 partitions and the number of producers was of our choice, we defined that the optimal number of producers should be also 6, to take advantage of the parallelism and augment the throughput.

The computation of the minimum and maximum temperature is done following a Voting Replication Tactic, meaning that the most common value between consumer groups will be the one accepted. When there is disagreement, that is, when the values given are all distinct, our strategy states that the temperature computed will be the minimum or the maximum of the given values, accordingly.

UC6

To minimize the possibility of reprocessing records in case a Kafka Consumer crashes, the consumer should be forced to commit the offset instead of committing periodically according to an `auto.commit.interval.ms`. To force the consumer to perform a commit, the property `enable.auto.commit` should be set to false, and the consumer should call the `commitSync` method after the processing. By doing this, in case a failure happens and the consumer crashes, it will start again in the record where the failure occurred.