Homework 5 (Extra Credit) | Streams

CSE 344 - Introduction to Data Management

You will find the starter files linked on the website as “cse344-hw5.zip”.

Note on collaboration: We encourage you to work in groups for these assignments. Discuss and argue your approach at a high level. You should type your own solutions (i.e. do not copy work).

While we provide a testing framework for components of your stream processing pipeline, the testing we provide is not an end-to-end test. It is up to you to print/debug your results to check if it makes sense.

Submit your solution files to Canvas. Things to turn in:

- hw5_code.zip

1. Kafka and Flink (20 points)

In this section, we will be processing the house power consumption data set provided by UC Irvine’s ML dataset repository. This dataset describes the minute-by-minute power consumption of a house for almost 4 years. For more details, refer to the schema description on the linked site.

To run your code on top of the Kafka system, you will have to set up a Kafka and ZooKeeper server. We have provided shell scripts for you to spin up a single Kafka server with the appropriate topics created and other tasks as well. These scripts merely call Kafka’s provided scripts in order to perform the setup. **We recommend that you do NOT run your server on attu as different instances of ZooKeeper and Kafka (from you and your peers who are also doing this assignment) may collide.**

Use the script **startCluster.sh** to spin up a ZooKeeper + Kafka instance and also create the topics.

Because Kafka persists data automatically, **you should clear the data before each test run of your program**. To do this, use the script **resetMessages.sh** to temporarily set the retention period of each topic to 1 second.

When you are done with this section you should shut down the server instance by using the script **stopCluster.sh**. This will delete the topics for this HW.

We have also provided the code in your starter file that will generate an input stream of rows of the input file. Those messages will be set to the topic “data_in”. The format for this row, and thus the serialization and deserialization technique is facilitated through Apache Avro (provided through Flink). Avro describes serialization through JSON descriptors. **For the input data schema, refer to the “hpc_avro.json” file and the “dataIn” method that converts the data file into a stream.**

Your job is to complete the Flink dataflow logic for the methods “partA” and “partB”. For these methods, you will be adding Flink operators to the execution environment. Note that these dataflows are done completely in Apache Flink, i.e. you do not need to worry about intermediate serialization and deserialization to Kafka. The serialization and deserialization processes from Kafka to Flink (and back to Kafka) have been provided.

The combined streaming system of Kafka and Flink will be using localhost (your local machine’s network) to relay messages. By default, Kafka consumers use at-most-once delivery guarantees and Flink uses at-least-once delivery guarantees internally. This means that **running your program over the 2-million+ data points may occasionally have different results between separate executions of your program.** Using the above default setup, you may note that ZooKeeper is running on its default port, 2181 and Kafka is running on its default port, 9092.

To test your solution’s operators use:

```
mvn clean compile
mvn test
```

To run your program’s main method use:

```
mvn clean compile
mvn exec:java -DmainClass="edu.uw.cs.HW5"
```

(a) (10 points) Find the “apparent power” of the house when “reactive power” is high. To do this, filter out rows that have a “global reactive power” less than or equal to 0.1 kilowatts. Then, compute the “global apparent power” for each record with the following formula:
“global apparent power” = ((“global active power”)^2 + (“global reactive power”)^2)$^{0.5}$

(b) (10 points) Find the average sub-meterings for every 24-hour window. Use a time-based tumbling window. The row you produce should have the start timestamp for each window you find and the respective average sub-meterings.

When your program finishes processing the data file, you will notice that the application hangs (does not terminate). This is due to the unbounded nature of our data as your program anticipates more data. At this point, you can check the cardinality of your processed results:

As a sanity check, you can observe the number of rows in the “data_in” topic with the below command:

```
./kafka_2.12-2.2.0/bin/kafka-console-consumer.sh \
  --bootstrap-server localhost:9092 \
  --topic data_in \
  --from-beginning
```

When the consumer appears to hang, interrupt the process with ctrl-c. You should get the message “**Processed a total of 2074863 messages**”. This is the number of rows in the original data.

To check the “part_a_out” topic use:

```
./kafka_2.12-2.2.0/bin/kafka-console-consumer.sh \
  --bootstrap-server localhost:9092 \
  --topic part_a_out \
  --from-beginning
```

You should see “**Processed a total of 1016186 messages**”

To check the “part_b_out” topic use:

```
./kafka_2.12-2.2.0/bin/kafka-console-consumer.sh \
  --bootstrap-server localhost:9092 \
  --topic part_b_out \
  --from-beginning
```

You should see “**Processed a total of 1441 messages**”