Guided Programming: Consumers, Multiple Channels, and Sliding Windows

44-671: Streaming Data

The same warnings that applied to notebooks and the Docker application from earlier assignments applies here.

When running a barbeque smoker, an accomplished (or even novice) pitmaster will closely monitor the temperatures of the cooking implement and the food to ensure everything is tasty. Often on long cooks, the following (less than favorable) events can happen:

- The temperature of the smoker can suddenly drop.
- The food hits a temperature where it is attempting to evaporate moisture, and it stays close to this temperature for an extended period of time (much like humans sweat to regulate temperature). We call this the "stall".

Modern technology has graced us with thermometers that allow us to track and record temperatures and keep a history of the temperatures of both the smoker and the food over time. We will be working with a data set generated from a thermometer that records the temperature *every thirty seconds*. The particular meal involved two different channels of data for food and the temperature of the smoker. We will be looking at sliding windows of data to detect the events described above. Specifically:

- If the temperature of the smoker drops more than 15 degrees in 2.5 minutes, we want to alert the pitmaster that the smoker needs attention
- If the temperature of the food does not change by more than one degree over a 15 minute time period, we want to report that the food has hit the stall

The producer code that reads the historical data has been provided.

1. Start your Docker application you set up in the Software Installation assignment
2. Open a web browser and open JupyterLab (`http://localhost:8888/lab`)
3. Upload the following files:
    - `bbq-producer.ipynb`
    - `smoker-temps.csv`
4. Change the appropriate line in the producer notebook with your RabbitMQ Docker instance name
5. Create a new Python Notebook named `yourname-consumer.ipynb` (replace `yourname` with your name)
6. Create a new cell in your consumer notebook that contains the following code (make sure to change the name of the RabbitMQ Docker instance!):

```python
import pika
import pickle
from collections import deque

# change 'streaming-data_rabbitmq_1' to the name of your rabbitMQ Docker instance
connection = pika.BlockingConnection(pika.ConnectionParameters('streaming-data_rabbitmq_1'))
channel = connection.channel()
queues = ['smoker', 'channel1', 'channel2']
for q in queues:
    channel.queue_declare(queue=q)

# Deal with a sliding window of time
smoker_window = deque(maxlen=5)
c1_window = deque(maxlen=30)
c2_window = deque(maxlen=30)
c1stall = False
c2stall = False

def smoker_callback(ch, method, properties, body):
    data = pickle.loads(body)
    smoker_window.append(data)
    if len(smoker_window) == smoker_window.maxlen and smoker_window[-1][1]-smoker_window[0][1] < -15:
        print(f'Smoker needs attention at time {smoker_window[-1][0]}')

def c1_callback(ch, method, properties, body):
    global c1stall
    data = pickle.loads(body)
    c1_window.append(data)
    if not c1stall and len(c1_window) == c1_window.maxlen and abs(c1_window[-1][1]-c1_window[0][1]) < 1:
        c1stall = True
        print(f'Channel 1 hit stall at time {c1_window[0][0]}')

def c2_callback(ch, method, properties, body):
    global c2stall
    data = pickle.loads(body)
    c2_window.append(data)
    if not c2stall and len(c2_window) == c2_window.maxlen and abs(c2_window[-1][1]-c2_window[0][1]) < 1:
        c2stall = True
        print(f'Channel 2 hit stall at time {c2_window[0][0]}')


channel.basic_consume(queue='smoker', auto_ack=True, on_message_callback=smoker_callback)
channel.basic_consume(queue='channel1', auto_ack=True, on_message_callback=c1_callback)
channel.basic_consume(queue='channel2', auto_ack=True, on_message_callback=c2_callback)
channel.start_consuming()
```

7. Create another cell in your consumer notebook with the code `connection.close()`
8. Run your consumer code (the first cell you created); fix errors that occur!
9. Run the producer code in the producer notebook
10. Add a new Markdown cell at the bottom of your producer notebook and answer the following questions:
    - What times did the smoker need attention?
    - What time did channel 1 hit its stall? Channel 2?
    - We used a sliding window approach; based on your reading of the Python Documentation (linked on the course website) how do you think we limited our code to only look at the last 2.5 minutes of smoker data and the last 15 minutes of food data?
    - Why do you think a sliding window approach was more appropriate than other windowing approaches (or considering the entire temperature history)? Do you think a different approach might be even better?
    - Why do you think we used different work queues for each channel of data?
11. "Run" your Markdown cell
12. Export your notebook as a PDF (make sure it contains your answers and the output from your code!) and submit the PDF to the course website.
13. Close your notebooks and stop Docker application; note there's no need to stop the producer (it knows when it is done generating data and stops itself).