Real time stream processing with Kafka - Part 2.2

Thanasak Harisombut

Date: 22/10/2020

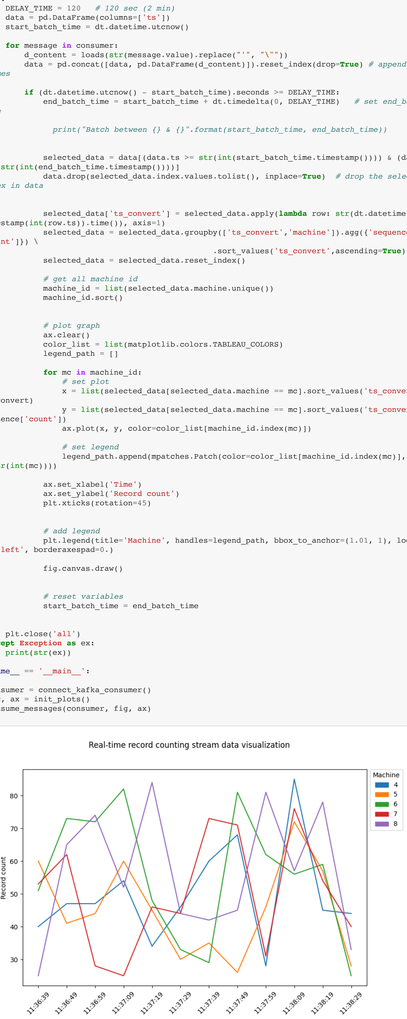
Version: 1.0

Environment: Python 3.7.4 and Jupyter notebook

2.2 Memory Event Consumer

In [1]:
```
# import statements
from time import sleep
from kafka import KafkaConsumer
import datetime as dt
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches

from json import loads
import pandas as pd

# this line is needed for the inline display of graphs in Jupyter Notebook
%matplotlib notebook

topic = 'Streaming_Linux_memory'


def connect_kafka_consumer():
    _consumer = None
    try:
        _consumer = KafkaConsumer(topic,
                                  consumer_timeout_ms=20000, # stop iteration if no message af
ter 10 sec
                                  auto_offset_reset='latest', # comment this if you don't want
to consume earliest available message
                                  bootstrap_servers=['localhost:9092'],
                                  value_deserializer=lambda x: loads(x.decode('ascii')),
                                  api_version=(0, 10))
    except Exception as ex:
        print('Exception while connecting Kafka')
        print(str(ex))
    finally:
        return _consumer

def init_plots():
    try:
        width = 9.5
        height = 6
        fig = plt.figure(figsize=(width, height)) # create new figure
        fig.subplots_adjust(hspace=0.8)
        ax = fig.add_subplot(111) # adding the subplot axes to the given grid position
        ax.set_xlabel('Time')
        ax.set_ylabel('Record count')
        fig.suptitle('Real-time record counting stream data visualization') # giving figure a t
itle
        fig.show() # displaying the figure
        fig.canvas.draw() # drawing on the canvas

        return fig, ax
    except Exception as ex:
        print(str(ex))
```

In [2]:
```
def consume_messages(consumer, fig, ax):
    try:
        DELAY_TIME = 120   # 120 sec (2 min)
        data = pd.DataFrame(columns=['ts'])
        start_batch_time = dt.datetime.utcnow()

        for message in consumer:
            d_content = loads(str(message.value).replace("'", "\""))
            data = pd.concat([data, pd.DataFrame(d_content)]).reset_index(drop=True) # append d
ataframes

            if (dt.datetime.utcnow() - start_batch_time).seconds >= DELAY_TIME:
                end_batch_time = start_batch_time + dt.timedelta(0, DELAY_TIME)   # set end_bat
ch_time

#                print("Batch between {} & {}".format(start_batch_time, end_batch_time))


                selected_data = data[(data.ts >= str(int(start_batch_time.timestamp()))) & (dat
a.ts < str(int(end_batch_time.timestamp())))]
                data.drop(selected_data.index.values.tolist(), inplace=True)  # drop the select
ed index in data


                selected_data['ts_convert'] = selected_data.apply(lambda row: str(dt.datetime.f
romtimestamp(int(row.ts)).time()), axis=1)
                selected_data = selected_data.groupby(['ts_convert','machine']).agg({'sequence
': ['count']}) \
                                                .sort_values('ts_convert',ascending=True)
                selected_data = selected_data.reset_index()

                # get all machine id
                machine_id = list(selected_data.machine.unique())
                machine_id.sort()


                # plot graph
                ax.clear()
                color_list = list(matplotlib.colors.TABLEAU_COLORS)
                legend_path = []

                for mc in machine_id:
                    # set plot
                    x = list(selected_data[selected_data.machine == mc].sort_values('ts_convert
').ts_convert)
                    y = list(selected_data[selected_data.machine == mc].sort_values('ts_convert
').sequence['count'])
                    ax.plot(x, y, color=color_list[machine_id.index(mc)])

                    # set legend
                    legend_path.append(mpatches.Patch(color=color_list[machine_id.index(mc)], l
abel=str(int(mc))))

                ax.set_xlabel('Time')
                ax.set_ylabel('Record count')
                plt.xticks(rotation=45)


                # add legend
                plt.legend(title='Machine', handles=legend_path, bbox_to_anchor=(1.01, 1), loc=
'upper left', borderaxespad=0.)

                fig.canvas.draw()


                # reset variables
                start_batch_time = end_batch_time


        plt.close('all')
    except Exception as ex:
        print(str(ex))

if __name__ == '__main__':

    consumer = connect_kafka_consumer()
    fig, ax = init_plots()
    consume_messages(consumer, fig, ax)
```




```
<ipython-input-2-442499b04a5b>:21: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/
indexing.html#returning-a-view-versus-a-copy
  selected_data['ts_convert'] = selected_data.apply(lambda row: str(dt.datetime.fromtimestamp(
int(row.ts)).time()), axis=1)
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-2-442499b04a5b> in <module>
     66     consumer = connect_kafka_consumer()
     67     fig, ax = init_plots()
---> 68     consume_messages(consumer, fig, ax)
     69

<ipython-input-2-442499b04a5b> in consume_messages(consumer, fig, ax)
      5         start_batch_time = dt.datetime.utcnow()
      6
----> 7         for message in consumer:
      8             d_content = loads(str(message.value).replace("'", "\""))
      9             data = pd.concat([data, pd.DataFrame(d_content)]).reset_index(drop=True) #
append dataframes

~/.local/lib/python3.8/site-packages/kafka/consumer/group.py in __next__(self)
   1190             return self.next_v1()
   1191         else:
-> 1192             return self.next_v2()
   1193
   1194     def next_v2(self):

~/.local/lib/python3.8/site-packages/kafka/consumer/group.py in next_v2(self)
   1198                 self._iterator = self._message_generator_v2()
   1199             try:
-> 1200                 return next(self._iterator)
   1201             except StopIteration:
   1202                 self._iterator = None

~/.local/lib/python3.8/site-packages/kafka/consumer/group.py in _message_generator_v2(self)
   1113     def _message_generator_v2(self):
   1114         timeout_ms = 1000 * (self._consumer_timeout - time.time())
-> 1115         record_map = self.poll(timeout_ms=timeout_ms, update_offsets=False)
   1116         for tp, records in six.iteritems(record_map):
   1117             # Generators are stateful, and it is possible that the tp / records

~/.local/lib/python3.8/site-packages/kafka/consumer/group.py in poll(self, timeout_ms, max_rec
ords, update_offsets)
    652         remaining = timeout_ms
    653         while True:
--> 654             records = self._poll_once(remaining, max_records, update_offsets=update_of
fsets)
    655             if records:
    656                 return records

~/.local/lib/python3.8/site-packages/kafka/consumer/group.py in _poll_once(self, timeout_ms, m
ax_records, update_offsets)
    699
    700         timeout_ms = min(timeout_ms, self._coordinator.time_to_next_poll() * 1000)
--> 701         self._client.poll(timeout_ms=timeout_ms)
    702         # after the long poll, we should check whether the group needs to rebalance
    703         # prior to returning data so that the group can stabilize faster

~/.local/lib/python3.8/site-packages/kafka/client_async.py in poll(self, timeout_ms, future)
    598                     timeout = max(0, timeout)  # avoid negative timeouts
    599
--> 600                 self._poll(timeout / 1000)
    601
    602             # called without the lock to avoid deadlock potential

~/.local/lib/python3.8/site-packages/kafka/client_async.py in _poll(self, timeout)
    630
    631         start_select = time.time()
--> 632         ready = self._selector.select(timeout)
    633         end_select = time.time()
    634         if self._sensors:

/usr/lib/python3.8/selectors.py in select(self, timeout)
    466                 ready = []
    467                 try:
--> 468                     fd_event_list = self._selector.poll(timeout, max_ev)
    469                 except InterruptedError:
    470                     return ready

KeyboardInterrupt:
```

In [ ]:
```

```