Streaming processing of cosmic rays using Drift Tubes detectors

Luca Agosti Nicola Bee Lorenzo Saccaro

7 September 2022

Università degli Studi di Padova

- The goal of our project is to simulate a live data processing network for a particle physics detector and show the results in a dashboard for continuous monitoring.
- The dataset is provided on a cloud storage bucket hosted on Cloud Veneto and is composed of multiple comma-separated values txt files.
- Our goal is to inject this data into a Kafka topic by emulating a continuous DAQ stream.

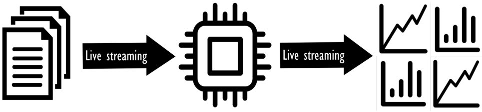


Figure: Very basic idea of live data streaming

Introduction (2/2)

Università degli Studi di Padova

- We processed our data using batch intervals varying from 1 to 5 seconds.
- After an initial data cleansing process all information extracted was wrapped in one message per batch and injected into a new Kafka topic.

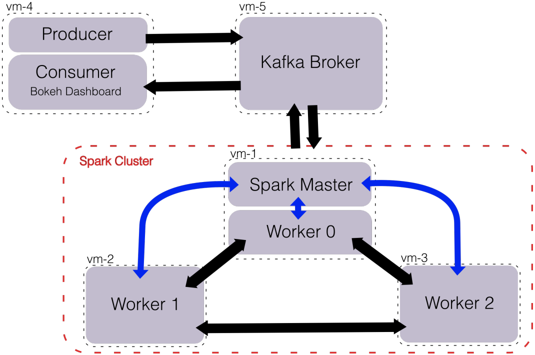
Tools

In order to implement our project different frameworks and python packages are used:

- Kafka 3.2.0: Distributed event streaming platform, used to manage the streaming of data, via *confluent_kafka* as interface
- Spark 3.3.0: Cluster computing framework for data analytics, used to perform distributed computation.
- Bokeh 2.4.3: Useful python package for interactive plotting

Network Architecture

- Our streaming-processing network architecture looks like the following:



Producer (1/2)

Define parameter

```python
from confluent_kafka import Producer
N_PARTITIONS = 12
BOOTSTRAP_SERVER = '10.67.22.61'
MSG_RATE = 1000 # number of messages per second
BATCH_FRACTION = 0.1 # can't be lower than 0.1
BATCH_SIZE = int(max(0.1*MSG_RATE, BATCH_FRACTION*MSG_RATE))
```

Create topic and producer

```python
def create_topics(admin, topics):
    new_topics = [NewTopic(topic, num_partitions=N_PARTITIONS, replication_factor=1)
                  for topic in topics]
    fs = admin.create_topics(new_topics, request_timeout=15.0)
    for topic, f in fs.items():
        try:
            f.result()  # The result itself is None
            print("Topic {} created".format(topic))
        except Exception as e:
            print("Failed to create topic {}: {}".format(topic, e))
producer = Producer({'bootstrap.servers':BOOTSTRAP_SERVER,
                     'linger.ms':20, 'batch.size':16384})
```

Producer (2/2)

UNIVERSITÀ DEGLI STUDI DI PADOVA

Connect to bucket

```
import boto3
url = 'https://cloud-areapd.pd.infn.it:5210'
s3_client = boto3.client('s3', endpoint_url=url, verify=False)
```

Write and send message

```
bucket_name = 'mapd-minidt-stream'
batch_count = 0 # counter for artificial delay

for key in s3_client.list_objects(Bucket=bucket_name)['Contents']:
    print('file:', key["Key"])
    # create line iterator
    line_reader = s3_client.get_object(Bucket=bucket_name,
                                       Key=key['Key'])['Body'].iter_lines()
    next(line_reader) # skip header line for each file
    for line in line_reader:
        producer.produce(topic_name, line) # produce message
        producer.poll(0) # pool producer
        batch_count += 1 # update counter
        if batch_count == BATCH_SIZE: # add artificial rate control
            time.sleep(BATCH_SIZE/MSG_RATE)
            batch_count = 0 # reset counter
producer.flush() # wait for last messages to be sent
```

Spark Analysis (1/3)

Università degli Studi di Padova

Define spark session builder

```
spark = SparkSession.builder \
   .master("spark://10.67.22.29:7077")\
   .appName("Test streaming")\
   .config('spark.executor.memory', '4g')\
   .config('spark.driver.memory', '1500m')\
   .config("spark.sql.execution.arrow.pyspark.enabled", "true")\
   .config("spark.sql.execution.arrow.pyspark.fallback.enabled", "false")\
   .config("spark.jars.packages","org.apache.spark:spark-sql-kafka-0-10_2.12:3.3.0")\
   .config("spark.eventLog.enabled", 'true')\
   .getOrCreate()
spark.conf.set("spark.sql.shuffle.partitions", 12)
```

Read data from Kafka

```
inputDF = spark.readStream.format("kafka")\
    .option("kafka.bootstrap.servers", KAFKA_BOOTSTRAP_SERVERS)\
    .option("kafkaConsumer.pollTimeoutMs", 4000).option('subscribe', 'data')\
    .option("startingOffsets", "latest").load()
```

Spark Analysis (2/3)

Università degli Studi di Padova

Message unpacking and pre-processing

```
# extract the value from the kafka message
csv_df = inputDF.select(col("value").cast("string")).alias("csv").select("csv.*")
# split the csv line in the corresponding fields
df = csv_df.selectExpr("cast(split(value, ',')[0] as int) as HEAD",
                       "cast(split(value, ',')[1] as int) as FPGA",
                       "cast(split(value, ',')[2] as int) as TDC_CHANNEL",
                       "cast(split(value, ',')[3] as long) as ORBIT_CNT",
                       "cast(split(value, ',')[4] as int) as BX_COUNTER",
                       "cast(split(value, ',')[5] as double) as TDC_MEAS"
# remove unwanted rows
df = df.filter(df.HEAD==2)
# add CHAMBER column for easier grouping later
df = df.withColumn("CHAMBER", \
               when((df.FPGA == 0)&(df.TDC_CHANNEL>=0)&(df.TDC_CHANNEL<64), 0)\
              .when((df.FPGA == 0)&(df.TDC_CHANNEL>=64)&(df.TDC_CHANNEL<128), 1)\
              .when((df.FPGA == 1)&(df.TDC_CHANNEL>=0)&(df.TDC_CHANNEL<64), 2)\
              .when((df.FPGA == 1)&(df.TDC_CHANNEL>=64)&(df.TDC_CHANNEL<128), 3)
                  )
# compute absolute time
df = df.withColumn("ABSOLUTE_TIME", 25*(df.ORBIT_CNT * 3564 +
                    + df.BX_COUNTER + df.TDC_MEAS/30))
```

Spark Analysis (3/3)

UNIVERSITÀ DEGLI STUDI DI PADOVA

Batch function

```
def batch_func(df, epoch_id):
  df.persist()
  hit_count = df.count()
  hit_count_chamber = df.groupby('CHAMBER').agg(count('TDC_CHANNEL')\
  .alias('HIT_COUNT')).sort("CHAMBER").select('HIT_COUNT')
  tdc_counts = df.groupby(['CHAMBER', 'TDC_CHANNEL'])\
  .agg(count('ORBIT_CNT').alias('TDC_COUNTS')).persist()
  ch0_tdc_counts = tdc_counts.filter(tdc_counts.CHAMBER==0).select('TDC_COUNTS')
  # [... similar for 1,2,3]
  hit_count_chamber = hit_count_chamber.toPandas().values.reshape(-1)
  ch0_tdc_counts_hist, ch0_tdc_counts_be = np.histogram(ch0_tdc_counts.toPandas()\
  .values.reshape(-1), bins = edges_list)
  # [... similar for 1,2,3]
  msg = { 'msg_ID': ID,
          'hit_count': hit_count,
          'hit_count_chamber': hit_count_chamber.tolist(),
          'tdc_counts_chamber': {
            '0': {
                'bin_edges': edges_list_to_print,
                'hist_counts': ch0_tdc_counts_hist.tolist()},
            # [... similar for 1,2,3] [...]
  producer.produce(TOPIC_NAME, json.dumps(msg).encode('utf-8'))
  producer.poll(0)

df.writeStream.outputMode("update").foreachBatch(batch_func)\
.trigger(processingTime='5 seconds').start().awaitTermination()
```

Consumer and Dashboard (1/2)

UNIVERSITÀ DEGLI STUDI DI PADOVA

Create a consumer

```
from confluent_kafka import Consumer
BOOTSTRAP_SERVER = '10.67.22.61'
consumer = Consumer({'bootstrap.servers': BOOTSTRAP_SERVER, 'group.id': 0})
consumer.subscribe(['results'])
```

Consumer and Dashboard (1/2)

Università degli Studi di Padova

Create a consumer

```
from confluent_kafka import Consumer
BOOTSTRAP_SERVER = '10.67.22.61'
consumer = Consumer({'bootstrap.servers': BOOTSTRAP_SERVER, 'group.id': 0})
consumer.subscribe(['results'])
```

Define an update function

```
def update():
  msg = consumer.poll(timeout=1)
  while msg == None: msg = consumer.poll(timeout=1)
  info = json.loads(msg.value())

  p_line.data_source.data['x'].append(info['msg_ID'])
  p_line.data_source.data['y'].append(info['hit_count'])
  p_line.data_source.trigger('data', p_line.data_source.data, p_line.data_source.data)
  # [...]
  hist0.data_source.data = {'top'  :info['tdc_counts_chamber']['0']['hist_counts'],
                            'left' :info['tdc_counts_chamber']['0']['bin_edges'][:-1],
                            'right':info['tdc_counts_chamber']['0']['bin_edges'][1:]
  hist0.data_source.trigger('data', hist0.data_source.data, hist0.data_source.data)
  # [... similar for 1,2,3] [...]
```

Consumer and Dashboard (2/2)

UNIVERSITÀ DEGLI STUDI DI PADOVA

Define the plots

```
p = figure(plot_width=700, plot_height=250, title="Total number of hits", ...)
p.x_range.follow = "end"
p.x_range.follow_interval = 50
p.x_range.range_padding = 0
p_line = p.line([], [], color="firebrick", line_width=3)
# [...]
h0 = figure(width=350, height=250, title = "Chamber 0", ...)
h_tick = FixedTicker(ticks=[0,10,20,30,40,50,60,70], minor_ticks=[5,15,25, ...])
h_overrides = {70: '>70'}
h0.xaxis.ticker = h_tick
h0.xaxis.major_label_overrides = h_overrides
hist0 = h0.quad(top=[], bottom=0, left=[], right=[], line_alpha=0, fill_color='blue')
# [...]
```

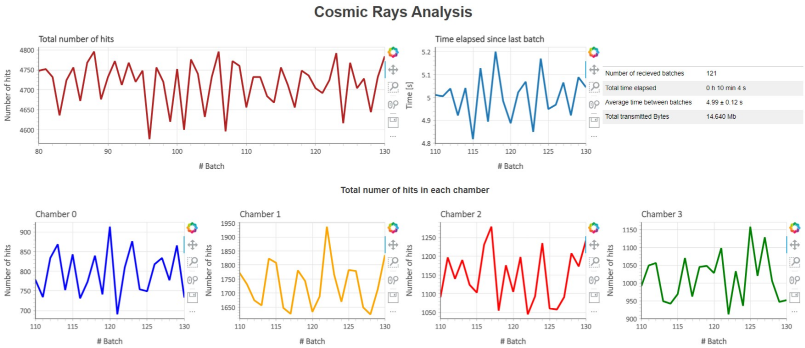
Define the plots

```
p = figure(plot_width=700, plot_height=250, title="Total number of hits", ...)
p.x_range.follow = "end"
p.x_range.follow_interval = 50
p.x_range.range_padding = 0
p_line = p.line([], [], color="firebrick", line_width=3)
# [...]
h0 = figure(width=350, height=250, title = "Chamber 0", ...)
h_tick = FixedTicker(ticks=[0,10,20,30,40,50,60,70], minor_ticks=[5,15,25, ...])
h_overrides = {70: '>70'}
h0.xaxis.ticker = h_tick
h0.xaxis.major_label_overrides = h_overrides
hist0 = h0.quad(top=[], bottom=0, left=[], right=[], line_alpha=0, fill_color='blue')
# [...]
```

Define the layout and create the callback

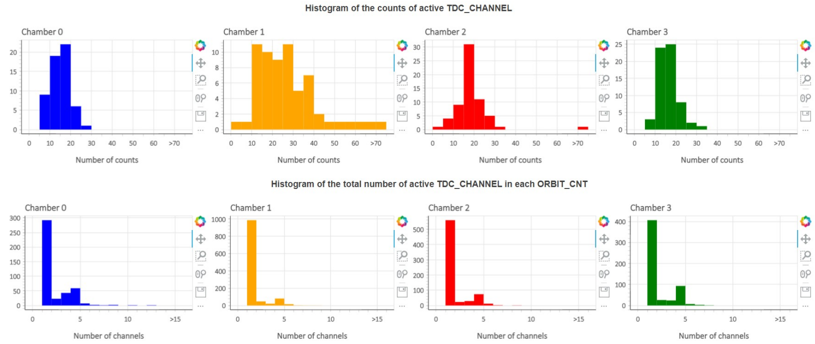
```
lay_out = layout([ [main_title], [p,t, info_table], [q_title], [q0, q1, q2, q3], ...
                   [h_title], [h0, h1, h2, h3], ... [d_title], [d0, d1, d2, d3] ])
curdoc().add_root(lay_out)
curdoc().theme = 'light_minimal'
curdoc().add_periodic_callback(update, 1)
```

Dashboard Result (1/3)



Dashboard Result (2/3)

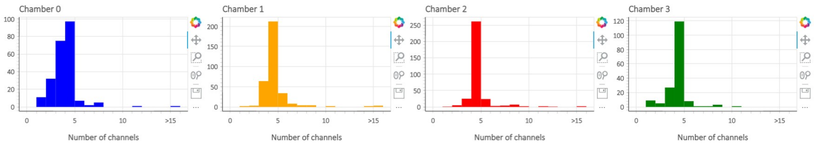
UNIVERSITÀ DEGLI STUDI DI PADOVA



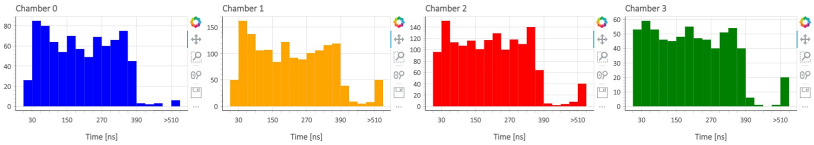
Dashboard Result (3/3)

Università degli Studi di Padova

Cumulative histogram of the counts of active TDC_CHANNEL, when scintillator register a signal



Cumulative histogram of the driftime



Performance analysis

We studied how the batch processing time scales by varying the following parameters:

Performance analysis

We studied how the batch processing time scales by varying the following parameters:

- Number of workers (1-3)

Performance analysis

We studied how the batch processing time scales by varying the following parameters:

- Number of workers (1-3)
- Number of partitions of data topic (1, number of cores, $2\times$ number of cores)

Performance analysis

We studied how the batch processing time scales by varying the following parameters:

- Number of workers (1-3)
- Number of partitions of data topic (1, number of cores, $2\times$ number of cores)
- Batch interval (1-5 seconds)

Initial transient

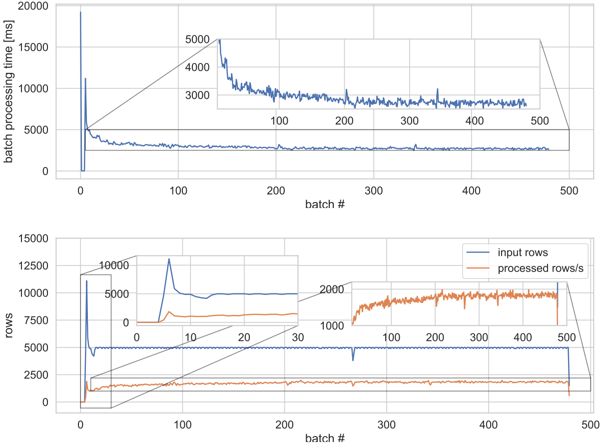


Figure: Processing time with 3 workers, 12 partitions and 1000 msg/s

Scaling

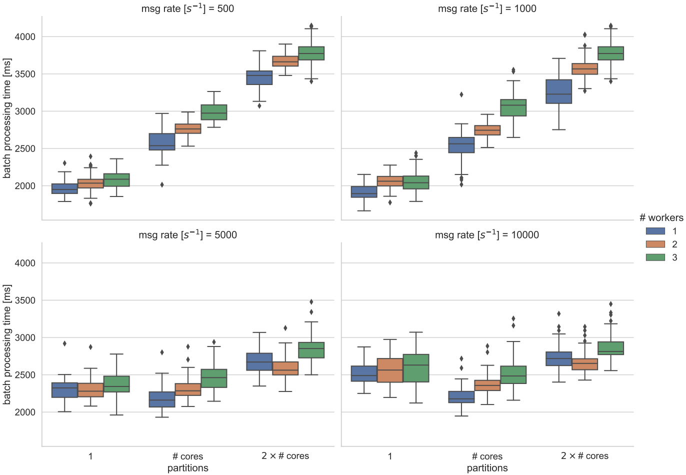


Figure: Processing time with batch interval of 5 seconds

Batch interval

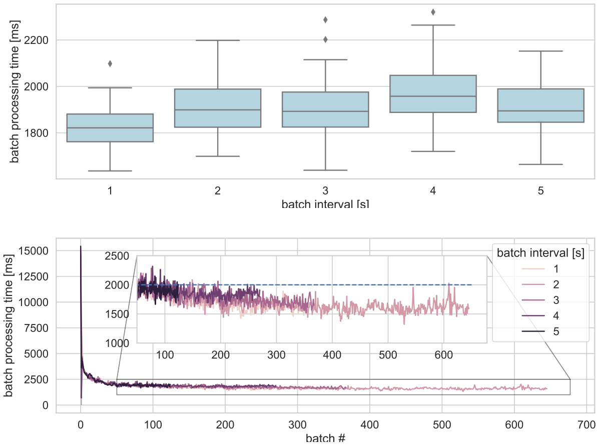


Figure: Processing time with 1 worker, 1 partition and 1000 msg/s

Final remarks

- With our implementation and cluster setup we are able to process batches with intervals between 3 and 5 seconds (even 2 seconds, with some caveats)

Final remarks

- With our implementation and cluster setup we are able to process batches with intervals between 3 and 5 seconds (even 2 seconds, with some caveats)
- Try with a different implementation that better fits the task assigned:

Final remarks

- With our implementation and cluster setup we are able to process batches with intervals between 3 and 5 seconds (even 2 seconds, with some caveats)
- Try with a different implementation that better fits the task assigned:
  - Parallel execution of queries each on a single partition

Final remarks

- With our implementation and cluster setup we are able to process batches with intervals between 3 and 5 seconds (even 2 seconds, with some caveats)
- Try with a different implementation that better fits the task assigned:
  - Parallel execution of queries each on a single partition
  - Write each result in a different topic