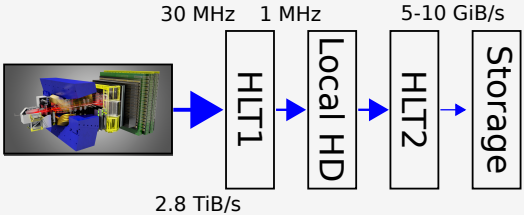
Scalable monitoring data processing for the LHCb software trigger

R. Aaij, R. Matev and S. Petrucci

On behalf of the LHCb RTA project

LHCb Run 3 Dataflow

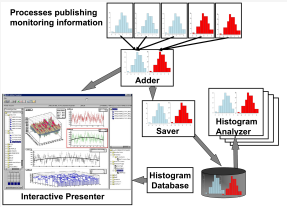


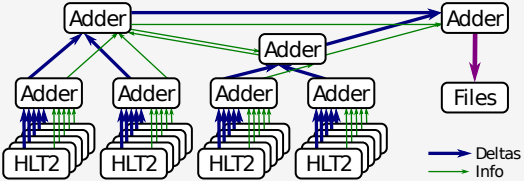
Scheme of the Run 3 Dataflow running from 2021 [1].

- The High Level Trigger (HLT) reduces background while maintaining high efficiency for physics signal candidates.
  - HLT1 reads the detector performing quick decision and writing on a local buffer.
  - HLT2 reconstruct the event performing a more detailed analysis.
- Everything rejected at this stage is permanently lost.
- A monitor system is fundamental to detect problems.
- A small fraction of the HLT computing resources is used to generating monitoring histograms and counters.

Current infrastructure

- HLT1:
  - Running with DIM as transport protocol.
  - Working with full histograms (messages are not split into deltas and info)[2].
- HLT2:
  - Running with ØMQ as messaging library.
  - Each node of the Event Filter Farm (EFF) is grouped into sub-farms and top level adders.
  - In order to lower the network traffic, two types of messages are created:
    1. Full messages with metadata
    2. Increments.
  - The adders merge the messages and publish them.





Current HLT2 implementation

- Completely customized system written using ØMQ as messaging library.
- Small resource consumption on the farm nodes.
- Adder tasks controlled by LHCb Experiment Control System (ECS).
- The queue has to be tuned requiring a simulation software:
  - Finding the optimal parameters of the queue is crucial in order to achieve the best performance from the system.
  - If the queue is not very well optimized, the system starts dropping messages.

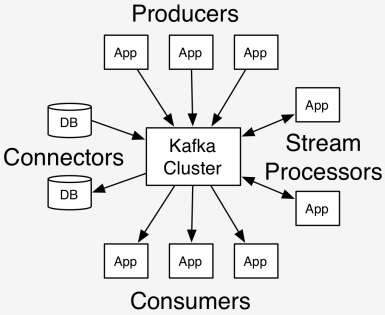
Apache Kafka

Apache Kafka is a distributed streaming platform:

- Publish and subscribe to streams of records.
- Store streams of records in a fault-tolerant way.
- Process streams of records as they occur.

Kafka is generally used for two broad classes of applications:

- Building real-time streaming data pipelines that reliably get data between systems or applications.
- Building real-time streaming applications that transform or react to the streams of data.



From Kafka official documentation

Why Kafka?

- Already used by LHCb Online for logs and monitoring [3].
- Commercial software under Apache license.
- Easier to implement/maintain.
- Queue optimization already performed.

Prototype's configuration

A Python prototype running with Confluent-Kafka was built.
It simulates the expected monitoring output of a HLT1 task.

Producer

Sending:

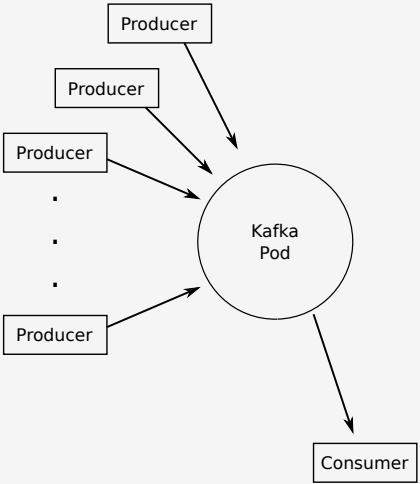
- 10KB/s ($\sim$ 1000 Counters).
- 100KB/10s ($\sim$ 100 1D histograms)
- 10MB/60s (2D and 1D histograms)

Consumer

- Checking for new messages.
- Emulating the memory access pattern of adding histograms.
- Deleting expired messages.

Prototype's configuration

- Kafka instance running on a Kubernetes' Pod.
- Several producers subscribe to the queue starting to push the fragments.
- A single consumer subscribes to the queue and consumes the messages, merging the fragments.



Prototype's goal

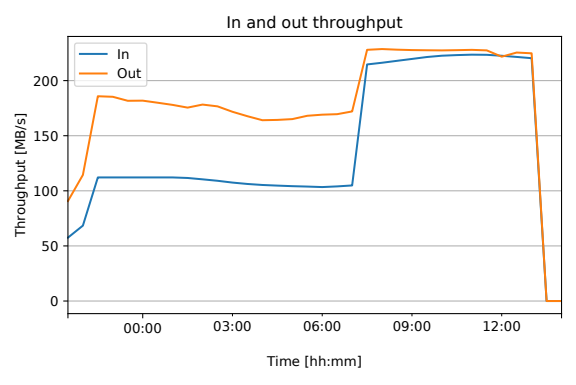
- Feasibility test with single consumer configuration (no multi-level adders).
  - Running using the existing LHCb Online infrastructure.
  - Simulating a realistic condition for HLT1 with few hundreds of producers.
- Stress test for a single consumer.
- Resilience to failures shutting down the consumer for few minutes.

Measurements with Grafana

The performance of the system was monitored using the Grafana interface already existing in LHCb Online.

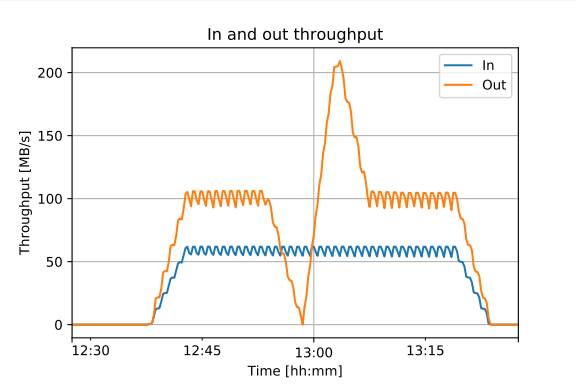
Results

Stability check:
203 and 406 Producers, 1 Consumer



Results

Failure test:
105 Producers, 1 Consumer



Conclusion

- A prototype of the HLT monitoring system using Kafka was implemented and tested.
- The prototype runs smoothly using the LHCb Online Kafka instance.
- The system showed effective recovery in response to a failure.

Future work

- Improving the performance porting the prototype to C++.
- Lowering the network load by adding more consumers.
- Performing additional tests to check the reliability.
- Building up a prototype for the HLT2 requirements.

Thank you for your attention

References

The LHCb collaboration
LHCb Upgrade Computing Model Technical Design Report, *CERN/LHCC 2018-14*

O.Callot et all.
Online Data Monitoring in the LHCb Experiment, *CHEP2007*

H. Mohamed
Deploying a "push" model Prometheus, *CHEP 2018*

R. Aaij
Reliable Monitoring for the LHCb HLT, ROOT workshop 2018

Some example code

Producer

```
from confluent_kafka import Producer

p = Producer({'bootstrap.servers': 'kube12.lbdaq.cern.ch:30401'}

while (True):
    p.poll(0)
    iterTime = time.time()
    if (iterTime - shortMsgTimer) > shortMsgFreq:
        shortMsgTimer = iterTime
        msgCount += 1
        timeBin = math.ceil(iterTime / shortMsgFreq) * shortMsgFreq
        payload = generatePayload(shortMsgSize)
        data = "{} {} {} {} {}".format(3, shortMsgFreq, processPid,
                                       timeBin, payload)
        p.produce("hlt1_one_partition_one_replication",
                  data.encode('utf-8'),
                  key=str(shortMsgSize),
                  callback=delivery_report)
    time.sleep(0.1)
p.flush()
```

Consumer

```
from confluent_kafka import Consumer, KafkaError

c = Consumer({
    'bootstrap.servers': 'kube12.lbdaq.cern.ch:30401',
    'group.id': 'mygroup'
})

c.subscribe(['hlt1_one_partition_one_replication'])

while True:
    msg = c.poll(0.1)

    if msg is None:
        continue
    if msg.error():
        print("Consumer error: {}".format(msg.error()))
        continue

    data = msg.value().decode('utf-8')
    id, payload, sourceId, timeBin, timeWindow = decodeMsg(data)
    msgSize = float(msg.key())
    timersKey = makeKey(id, timeBin, timeWindow)

    if not timersKey in timers:
        timers[timersKey] = time.time()
    for key in fullValues.keys():
        if dumpData(timeStart, int(key.split("-")[2])):
            delKeyList.append(key)
    for key in delKeyList:
        if key not in fullValues:
            print("Missing key")
            print(fullValues.keys())
        try:
            del fullValues[key]
        except KeyError:
            print("Key {} not found".format(key))
    delKeyList = []

    mergePayload(fullValues, id, payload, timeBin, timeWindow)

c.close()
```