Developing PulsarIO Connector

By Marco Robles
Agenda

- Introduction
- What is Pulsar?
- Initial approach
- Current implementation
- Example
- Next steps
- Q&A
Who am I?

Software Engineer
@ Wizeline
Who we are

Wizeline, a global technology services provider, builds high-quality digital products and platforms that accelerate time-to-market.

- **We focus on measurable outcomes**, partnering with our customers to modernize core technologies, mature data-driven capabilities, and improve user experience.

- **Our adaptive teams** provide the right combination of solutions, capabilities, and methodologies to deliver results, while partnering with our customers’ teams to foster innovation through continuous learning.

- **We are invested in doing well while doing good**, striving to make a positive impact where we live and work. Our diverse culture of innovation, ownership, and community, combined with our Academy, creates an inspiring environment for talent to build long-term careers.
Wizeline delivers seamless, scalable digital solutions, embedding the right technology, methodology, and mindsets within our customers’ organizations. Our technology expertise and focus on AI & continuous learning, combined with our diverse and inclusive teams, allow us to deliver what you need right now, while also building a roadmap to your future.

20+ nationalities represented at Wizeline globally

2000+ Wizeline employees
Wizeline Team - Beam's Contributors

Benjamin Gonzalez
Mike Hernandez
Fernando Morales
Daniela Martin
Andoni Guzman
Elias Segundo
Lupita Amezcua

Austin, 2022
What is Pulsar?

- Introduction
- **What is Pulsar?**
- Initial approach
- Current implementation
- Example
- Next steps
- Q&A
The basics

A pub/sub messaging system originally catered towards queuing use cases
What is Apache Pulsar?

A distributed messaging and streaming platform originally created at Yahoo.

Pulsar is a multi-tenant, high-performance solution for server-to-server messaging.
Why Pulsar?

Unified Messaging Model
Simplify your data infrastructure and enable new use cases with queuing and streaming capabilities in one platform.

Multi-tenancy
Enable multiple user groups to share the same cluster, either via access control, or in entirely different namespaces.

Scalability
Decoupled data computing and storage enable horizontal scaling to handle data scale and management complexity.

Geo-replication
Support for multi-datacenter replication with both asynchronous and synchronous replication for built-in disaster recovery.

Tiered storage
Enable historical data to be offloaded to cloud-native storage and store event streams for indefinite periods of time.
Pulsar architecture
Pulsar messaging

Sequence ID  Message ID  Publish Time

https://pulsar.apache.org/docs/concepts-messaging
Sequence ID

Each Pulsar message belongs to an ordered sequence on its topic.
Assigned by the producer (optional)

Constraints:
- $\text{sequenceID} \geq 0$
- $\text{sequenceID}(N+1) > \text{sequenceID}(N)$
- It's not necessary for sequence IDs to be consecutive. There can be holes between messages.
Message ID

Indicates a message's specific position in a ledger and is unique within Pulsar cluster.

Constraints:
- It is not a numeric value.
- It has its own value type (Message ID class).
Publish time

The timestamp of when the message is published.

Automatically applied by the producer.
Consumer interface

Pulsar automatically manages topic cursors

Consumer begins reading at most recent unacked message
Reader interface

Applications manually control topic cursors

Reader begins reading at an application-specified message
Initial approach

- Presentation
- What is Pulsar?
- **Initial approach**
- Current implementation
- Example
- Next steps
- Q&A
A basic *splittable DoFn (SDF)* implementation:

1. Pair each $A$ with initial Restriction ($R$)
2. Split each $R$ to parallelize processing of $A$
3. Process element $A$ and $R_x$
4. Checkpoint / Split

@InitialRestriction

@SplitRestriction
Initial Pulsar splittable DoFn implementation

Which restriction can we use?

\[(\text{element, restriction}) \rightarrow (\text{element, restriction}_1) + (\text{element, restriction}_2)\]
Restriction?

ReadFromPulsarDoFn (topic, [0, inf))

ReadFromPulsarDoFn (topic, [0, 100))

ReadFromPulsarDoFn (topic, [100, 150))

ReadFromPulsarDoFn (topic, [150, inf))
Pulsar messaging

https://pulsar.apache.org/docs/concepts-messaging
In Kafka

Partition 0
 offsets inf
 0 1 2 3 4 5 ...

Partition 1
 offsets inf
 0 1 2 3 4 5 ...

Partition 2
 offsets inf
 0 1 2 3 4 5 ...

TOPIC
In Pulsar

Message ID

- Ledger ID
- Entry ID
- Bach-index
```java
public static final long getOffset(MessageId messageId) {
    MessageIdImpl msgId = (MessageIdImpl) messageId;
    long ledgerId = msgId.getLedgerId();
    long entryId = msgId.getEntryId();
    // Combine ledger id and entry id to form offset
    // Use less than 32 bits to represent entry id since it will get
    // rolled over way before overflowing the max int range
    long offset = (ledgerId << 28) | entryId;
    return offset;
}

public static final MessageId getMessageId(long offset) {
    // Demultiplex ledgerId and entryId from offset
    long ledgerId = offset >>> 28;
    long entryId = offset & 0x0F_FF_FF_FFL;
    return new MessageIdImpl(ledgerId, entryId, -1);
}
```

new MessageIdImpl( ledgerId, entryId, batchIndex );

( ledgerId, entryId, batchIndex )

Current message
(10, 5, 100)

175921860464740

Next message
(11, 0, 0)

193514046488576

17,592,186,023,836 > 32 bits (4,294,967,296)
What can we do?
Publish time

https://pulsar.apache.org/docs/concepts-messaging
Which client interface use?
Client interface

Consumer interface
Pulsar automatically manages topic cursors

Reader interface
Applications manually control topic cursors

Consumer begins reading at most recent unacked message
Reader begins reading at an application-specified message
Reader interface

Applications manually control topic cursors

Reader begins reading at an application-specified message
Current implementation

- Presentation
- What is Pulsar?
- Initial approach
- **Current implementation**
- Example
- Next steps
- Q&A
Publish time

- Sequence ID
- Message ID
- Publish Time

https://pulsar.apache.org/docs/concepts-messaging
Restriction

ReadFromPulsarDoFn (topic, [0, inf])

ReadFromPulsarDoFn (topic, [0, 1654111383825L])

ReadFromPulsarDoFn (topic, [1654111383825L, 1654111384289L])

ReadFromPulsarDoFn (topic, [1654111384289L, inf])
ReadFromPulsarDoFn

Splittable DoFn
class SourceDescriptor { String topic; long startOffset; Message messageRecord }

@GetInitialRestriction
OffsetRange initialRestriction(sourceDescriptor) {
    long startTime = 0;
    long endTime = Long.MAX_VALUE;
    if ( sourceDescriptor.startOffset != null ) {
        startTime = sourceDescriptor.startOffset;
    }
    if ( sourceDescriptor.endOffset != null ) {
        endTime = sourceDescriptor.endOffset;
    }
    new OffsetRange(startTime, endTime);
}
@ProcessElement

ProcessContinuation processElement(
    @Element SourceDescriptor sourceDescriptor,
    OffsetRangeTracker<OffsetRange, Long> tracker,
    OutputReceiver<PulsarMesasge> output) {

    // A reader is created from PulsarClient defining the starting point from the earliest available message in the topic.
    try {
        Reader<byte[]> reader = newReader(client, sourceDescriptor.topic);
        // The current processElement() call must respect the supplied restriction.
        // The restriction is [starting offset, infinity) - seek to it.
        reader.seek(tracker.getFrom());
        while (true) {
            Message message = reader.getNext();
            long currentTimestamp = message.getPublishTime();
            // if tracker.tryClaim() return true, sdf must execute work otherwise doFn must exit processElement() without doing any work associated or claiming more work
            if (!tracker.tryClaim(currentTimestamp)) {
                return ProcessContinuation.stop();
            }
        }
    }
}

Split restriction

ReadFromPulsarDoFn (topic, [0, inf])

ReadFromPulsarDoFn (topic, [0, 1654111383825L])

ReadFromPulsarDoFn (topic, [1654111383825L, 1654111384289L])

ReadFromPulsarDoFn (topic, [1654111384289L, inf])

Split restriction
@NewTracker

```java
OffsetRangeTracker newTracker(OffsetRange range) {
    // Since Pulsar is a streaming-unbounded process
    // User could define a bounded process or unbounded process on tracker
    if (restriction.getTo() < Long.MAX_VALUE) {
        return new OffsetRangeTracker(range);
    }
    // If user don't define a end range, it will continue calculating the range
    // with [currentRestrictionFrom, latestMessageInTopic), using
    // Pulsar Admin Client to retrieve the latest message available in topic
    return new GrowableOffsetRangeTracker(
        restriction.getFrom(),
        new GrowableOffsetRangeTracker.RangeEndEstimator() {
            long estimate() {
                return admin().latestMessageInTopic();
            }
        });
}
```
ReadFromPulsarDoFn (topic, [0, 1654111383825L])

ReadFromPulsarDoFn (topic, [1654111383825L, 1654111384289L])

0 1654111383825L

1654111383825L 1654111384289L
Timestamp observing

\[ \downarrow \]

Timestamp of each record

External clock observing

\[ \downarrow \]

Timestamp not associated
ReadFromPulsarDoFn
(topic, [0, 1654111383825L])

timestamp

current_timestamp
Watermark estimator

There are some build-on watermark estimator implementations in Java:

1. Manual
2. Monotonically increasing
3. Wall time
ReadFromPulsarIO has two types of timers:

- **Publish time**
- **Processing time**
```java
@ProcessElement
ProcessContinuation processElement(
    @Element SourceDescriptor sourceDescriptor,
    OffsetRangeTracker<OffsetRange, Long> tracker,
    OutputReceiver<PulsarMessage> output) {

    PulsarMessage pulsarMessage =
        new PulsarMessage(message.getTopicName(),
                           message.getPublishTime(),
                           message);

    Instant outputTimestamp = extractOutputTimestampFn.apply(message);

    output.outputWithTimestamp(pulsarMessage, outputTimestamp);
}
```
static class ExtractOutputTimestampFn {
    public static SerializableFunction<Message<byte[]>, Instant> 
        useProcessingTime() {
        return record -> Instant.now();
    }

    public static SerializableFunction<Message<byte[]>, Instant> 
        usePublishTime() {
        return record -> new Instant(record.getPublishTime());
    }
}
Example

- Presentation
- What is Pulsar?
- Initial approach
- Current implementation
- Example
- Next steps
- Q&A
PulsarIO Reader

```java
PulsarIO.Read reader = PulsarIO.read()
    .withClientUrl("pulsar_client_url")
    .withPulsarClient(SerializableFunction...)
    .withAdminUrl("pulsar_admin_url")
    .withTopic("topic")
    .withStartTimestamp(startTime)
    .withEndTimestamp(endExpectedTime)
    .withPublishTime();

pipeline.apply(reader);
```
PulsarIO Reader

```java
PulsarIO.Read reader = PulsarIO.read()
    .withClientUrl("pulsar_client_url")
    .withPulsarClient(SerializableFunction...)
    .withAdminUrl("pulsar_admin_url")
    .withTopic("topic")
    .withStartTimestamp(startTime)
    .withEndTimestamp(endExpectedTime)
    .withPublishTime()
    .withProcessingTime();

pipeline.apply(reader);
```
PulsarIO Writer

```java
PulsarIO.Write writer = PulsarIO.write()
    .withClientUrl("pulsar_client_url")
    .withTopic("topic");

List<byte[]> messages = new ArrayList<>();
messages.add("MESSAGE_1").getBytes());
messages.add("MESSAGE_2").getBytes());

pipeline.apply(Create.of(messages))
    .apply(writer);
```
Next steps

- Presentation
- What is Pulsar?
- Initial approach
- Current implementation
- Example
- Next steps
- Q&A
A lot work to do...

- Acknowledge messages
- Multi-topic partition
- Set a dynamic stop limit for reader and writer
- Allow subscription types
- ...

Thanks

Questions?

marco.robles@wizeline.com | dev@beam.apache.org
linkedin.com/in/marcoantoniorob
github.com/MarcoRob