Adam Bellemare - Building Event-Driven Microservices

6. Deterministic Stream Processing

3 key questions

How to choose the order of events that are coming from multiple partitions?

How to handle out-of-order and late-arriving events?

How to produce deterministic results when processing streams in near-real time VS when processing from the beginning of the streams?

Determinism with Event-Driven Workflows

Workflow is explicitly nondeterministic if based on

wall-clock

query external services

Fully deterministic processing is the ideal case

the reality is that our services can only achieve a BEST EFFORT determinism

Things we need to facilitate determinism

consistent timestamps

well-selected event keys

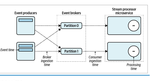
partition assignment

event scheduling

strategies to handle late-arriving events

Key concepts

Timestamps



There are 4 timestamps

Event time

Broker ingestion time

Consumer ingestion time

Processing time

Synchronizing Distributed Timestamps

Two independent systems cannot have PRECISELY the same system-clock time

They can only be NEARLY in sync

which is good enough for most cases

Network Time Protocol (NTP) servers

+/- few mS when sync inside cloud to +/- 100mS when sync with open Internet

for most cases they provide sufficient consistency

Processing with Timestamped Events

You cannot process events using naive (e.g. round-robin) order!

You should consider events' timestamps

Event Scheduling and Deterministic Processing

Your need EVENT SCHEDULING if the order of events matters to the business logic

Custom Event Schedulers

Some streaming frameworks allow to implement custom event schedulers

But becareful: many custom schedulers are nondeterministic in nature

Which time to use?

1. Usually the best is to use locally assigned EVENT TIME

provided you can rely on its accuracy

2. Next good choice is BROKER INGESTION TIME

If the producer has unreliable timestamps (and you can't fix it)

it is rare case

Timestamp Extraction (by Consumer)

is used to extract the right timestamp from the event (at consumer ingestion time)

Request-Response Calls to External Systems

may introduce nondeterministic results

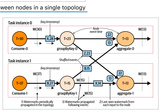
Watermarks

Watermark - is a declaration to downstream nodes within the same processing topology that all events of time t and prior have been processed

It does not affect the event scheduling of the node

it simply notifies the node that it should consider any events with a timestamp earlier than the water mark to be considered late

Watermark propagation between nodes in a single topology



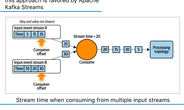
Watermark propagation between nodes in a single topology with multiple processors

Parallel Processing

The node's event time is the minimum of all of its input sources' event times

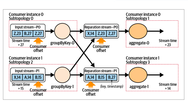
Stream Time

this approach is favored by Apache Kafka Streams



Stream time when consuming from multiple input streams

The next event to be processed is the smallest value of the two input buffers



Shuffling events via a repartition event stream

sends the repartitioned events back to the event broker using what's known as an internal event stream

Parallel Processing

Watermarking strategies can also use repartition event streams

in Apache Samza

Out-of-order events in an event stream partition

Problems and dealing with them

Out-of-Order and Late-Arriving Events

Out-of-order events in an event stream partition

Bounded vs Unbounded datasets

Bounded data sets (such as historical data processed in batch) are typically fairly resilient to out-of-order data: e.g. nightly batch processing

But this comes at the expense of high latency

Unbounded data sets

must consider the requirements of latency and determinism

"Late" is only a perspective of a specific consumer

Other consumers may consider this event as good

Watermarks

The event t' is considered late when it arrives after the watermark W(t). It is up to the specific node how to handle this event.

Stream time

The event t' is considered late when it arrives after the stream time has been incremented past t'. It is up to each operator in the subtopology how to handle this event.

Causes and Impacts of Out-of-Order Events

Sourcing from out-of-order data

Shuffling events via a repartition event stream

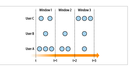
Multiple producers to multiple partitions

A single-threaded producer will not create out-of-order events in normal operation unless it is sourcing its data from an out-of-order source

Time-Sensitive Functions and Windowing

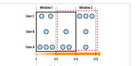
Windowing

grouping events together by time



Tumbling windows

a window of a fixed size



Sliding windows

a fixed window size and incremental step known as the window slide

Session windows

a dynamically sized window

3 main types of event windows

Handling Late Events

Strategy for handling out-of-order and late-arriving events should be determined at business level first!

Critical events such as financial transactions and system failures

may be required to be handled regardless of their position in the stream

Measurement style events, such as temperature or force metrics

may simply be discarded as no longer relevant

Ways

Drop event

Simply drop the event

The window is closed, and any time-based aggregations are already complete

Wait

Delay output of the window results until a fixed amount of time has passed

This incurs higher determinism

at the expense of increased latency

Grace period

Output the windowed result as soon as the window is deemed complete

Then, keep the window(s) around and available for the predetermined grace period

Whenever a late event arrives for that window, update the aggregation and output the newly updated aggregation

This is similar to the wait strategy, except updates are generated as late events arrive

Questions to determine good guidelines

How likely are late events to occur?

How long does your service need to guard against late events?

What are the business impacts of dropping late events?

What are the business benefits of waiting a long time to capture late events?

How much disk or memory does it take to maintain state?

Do the expenses incurred in waiting for late events outweigh the benefits?

Reprocessing vs Processing in Near-Real Time

During reprocessing (rewind and replay) the order should be the same

Steps

1. Determine the starting point

2. Determine which consumer offsets to reset

3. Consider the volume of data

ensure I/O quotas are ok

4. Consider the time to reprocess

scale the number of consumers if needed

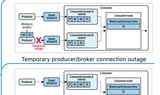
5. Consider the impact

e.g. you shold not re-email users

Intermittent Failures and Late Events

Producer/Event Broker Connectivity Issues

Normal operation prior to producer/broker connection outage



Temporary producer/broker connection outage

The producer is able to reconnect and publish its temporarily delayed events, while the consumer has already incremented its event time

Further Reading

"Streaming 101: The world beyond batch" by Tyler Akidau

https://github.com/ideordie/ideordie-digest/issues/66

"Distributed systems for fun and profit" by Mikito Takada

http://book.mixu.net/distsys/