Running Notes
==========

append
complete
update

In this session we will talk about generating the input data

Json data...

- Complete output mode

- update mode

orders
customers
order_items

/public/retail_db

we need to join
orders
customers
order_items

Running Notes
==========

Databricks

Delta format - Complete
table

Parquet, CSV - Complete mode does not work
table

========

Update Mode

==========

toTable will work with append mode, complete mode

with update mode it wont work

target table (final result table) t orders_final_result

source table (mirco batch table) s orders_result

if t.customer_id == s.customer_id

when matched then

update all the columns...

when not matched then

insert *

target table - 100000 records

source table - 2 inserts and 1 update

merge

==========

We have covered
- append
- update
- complete

readStream

Processing (Streaming Transformations)

writeStream

2 categories of transformations
==========

Stateless vs Stateful

Stateless - does not have to maintain the previous state
select, filter etc...

Stateful - here the state has to be maintained..
groupBy, join, windowing functions

State store holds the aggregation results for the previous micro batches (executor memory)

Amazon - 300+ million active customers

state store has to keep 300 million entries in the memory

unbounded - its a heavy operation
monthly - bounded (1 month interval)

1 million people make a purchase (still feasible to be kept in state store)

unbounded - mostly the data would be more (you might have to use your own solution)

bounded by time - weekly, monthly, quarterly (we can most probably use the state store)

=======

Triggers
=========

Microbatch

how big is a microbatch (how is this decided)

when is the new microbatch triggered.

Triggers

- unspecified (default)
once the first microbatch is complete it will immediately trigger the next microbatch

inputfolder - file1,file2,file3,file4

then we run the spark streaming application

microbatch1 - file1,file2
microbatch2 - file3 (currently in processing)
microbatch3 - file4

you want to process the files as soon as possible.

- fixed Interval (each microbatch will start after a certain time)

5 mins as the fixed interval

prev microbatch is completed in 2 mins...

it will wait for 3 more mins..

if the prev microbatch completes in 8 mins...

it will instantly trigger the next microbatch after completion of prev microbatch.

where we wish to collect a good sizeable amount of data and then process.

Triggers are applicable to the writeStream

.trigger(processingTime='10 seconds')

- Available Now (process the microbatch and then stop by itself)

.trigger(availableNow=True)

file1,file2,file3

it will process all the files

and then will stop

file4,file5

automatically handles incremental processing

========

how is a stream processing different than batch

the challenges involved with streaming

microbatch approach

Sources and the Sinks - socket, file, kafka...

console,file, delta, kafka

read
process
write

checkpoint directory - state, it maintains info on what all is processed.

3 output modes
- append
- complete
- update

State store

foreachBatch (custom logic)

how to avoid using the state store and implement our own logic.

Types of triggers

- unspecified (default)
- fixed interval
- available now

what is yet to be covered
==========

fault tolerance and exactly once guarantee

2 types of aggregations
- time bound aggregations (window aggregations)
    - tumbling window
    - sliding window
- continuous aggregations

concept of watermark for late arriving records..

Streaming joins

fault tolerance and exactly once guarantee
==========

Ideally a streaming applications should run forever.

It might stop..

1. Exception
2. Maintainance activities

our application should be able to stop and restart gracefully.

This means to maintain exactly once semantics..

1. Do not miss any input record
2. Do not create duplicate output records

Spark structured streaming provides ample support for this.

It maintains state of the microbatch in the checkpoint location

checkpoint location helps you to achieve fault tolerance

checkpoint location mainly contains 2 things -

1. read positions - which all files are processed.
2. state information - running total

Spark structured streaming maintails all the info it requires to restart the unfinished microbatch.

To guarantee exactly once semantics , 4 requirements should be met..

1. restart the application with the same checkpoint location - lets say in 3rd microbatch we got an exception.

in commits we would have 2 commits..

2. use a replayable source - consider there are 100 records in the 3rd microbatch

after processing 30 records it gave some exception.

these 30 records should be available to you when you start reprocessing.

when we use socket source then we cannot get older data back again.

kafka, file source both of them are replayable.

3. use deterministic computation - consider there are 100 records in the 3rd microbatch

after processing 30 records it gave some exception.

when we start reprocessing these 30 records , it should give the same output.

dollars_to_INR(10) - non deterministic

sqrt(4) - deterministic

4. use an idempotent sink - consider there are 100 records in the 3rd microbatch

after processing 30 records it gave some exception.

we are procesing 30 records 2 times?

we are writing this to output 2 times..

2nd time when you are writing the same output it should not impact us.

either it should discard the 2nd output or it should overwrite the 1st output with the 2nd one.

Aggregations are of 2 types
==========

1. continuous aggregation -
you purchase some grocery from a retail store

on purchase of 100 Rs you get 1 reward point.

if these reward points never expire (unbounded)

100 million customers..

every month they are getting 1 million new customers...

history in the state store will keep on growing.

In such cases its better to implement custom solution.

2. time bound aggregations (windowing aggregations)

one month window. the reward points expire after one month.

state store cleanup will take place each month.

There are 2 kind of windows
==========

1. Tumbling window - a series of fixed size, non overlapping time interval.
10 - 10:15
10:15 - 10:30
10:30 - 10:45
10:45 - 11

2. Sliding window - a series of fixed size, overlapping window.
10 - 10:15
10:05 - 10:20
10:10 - 10:25

The aggregation windows are based on event time and not on trigger time.

event time is the time that event is generated..

later this event will reach to us for processing and that time is called as trigger time.

{"order_id":57012,"order_date":"2020-03-02 11:05:00","order_customer_id":2765,"order_status":"PROCESSING", "amount": 200}

event time is 11:05

this event might reach to spark lets say at 11:20

amount of sales every 15 minutes..

Tumbling window

11 - 11:15
11:15 - 11:30
11:30 - 11:45
11:45 - 12

nc -lk 9970

{"order_id":57012,"order_date":"2020-03-02 11:05:00","order_customer_id":2765,"order_status":"PROCESSING", "amount": 200}

2020-03-02 11:00:00 | 2020-03-02 11:15:00| 900
2020-03-02 11:15:00 | 2020-03-02 11:30:00| 1500
2020-03-02 11:30:00 | 2020-03-02 11:45:00| 500
2020-03-02 11:45:00 | 2020-03-02 12:00:00| 400

To deal with late coming records we have a concept of watermark

what is a watermark?

- it is like setting an expiry date to a record.

The business says that we are looking for 99.9% accuracy.

99.9% of your records are never late than 30 minutes.

than you can set your watermark to 30 minutes.

1000 events... 1 event can arrive later than 30 minutes.

999 events will arrive before 30 minutes.

watermark boundary - 11:18

2020-03-02 11:15:00|2020-03-02 11:30:00| 1500|
2020-03-02 11:30:00|2020-03-02 11:45:00| 500|
2020-03-02 11:45:00|2020-03-02 12:00:00| 400|

remember that

1. watermark is the key to clean our state store.
2. events within the watermark are taken - this is guaranteed.
3. events outside the watermark may or may not be taken.

- complete
- update
- append

complete output mode will not allow us to clean the state store.

append mode

watermark with windowing aggregations

append mode we use watermark 30 mins.. (windowing aggregations)

watermark boundary 11:20

11 - 11:15

the window information will only be printed once this window expires.

Sliding window

11 - 11:15
11:05 - 11:20
11:10 - 11:25

Streaming Joins

==========

Structured streamig support 2 kind of joins

1. streaming df to static df

2. streaming df to another streaming df

streaming df to static df
==========

stream enrichments

lets take an example from banking domain..

I swipe a credit card at pos terminal..

whenever I swipe my card a transaction will be generated.

{"card_id":5572427538311236,"amount":5358617,"postcode":41015,"pos_id":
970896588019984,"transaction_dt":"2018-10-21 04:37:42"}

json data...

enrich the above stream

member_id, country, state, card_released_time

card_id,member_id,card_issue_date,country,state
5572427538311236,976740397894598,2014-12-15 08:06:58.0,United States,Tonawanda
5134334388575160,978465390240911,2012-10-17 11:55:14.0,United States,Kankakee
5285400498362679,978786807247400,2014-01-08 03:31:52.0,United States,Fallbrook

Left outer
==========

all the matching records + all the non matching records from the left table padded with nulls on the right.

is left outer join possible
left table is a static
right table is a stream

static stream

123

stream static

123

123, Null

left outer join to work

left side should be a stream and right side should be static.

right outer join to work

right side should be a stream and left side should be static.

AnalysisException: RightOuter join with a streaming DataFrame/Dataset on the left and a static DataFrame/DataSet on the right not supported;