DATA+AI SUMMIT BY databricks

BUILDING METRICS STORE WITH INCREMENTAL PROCESSING

Hang Li
June 2024

About Us

Instacart Ads Measurement Team

instacart

Soom Foods Reaches New Customers and Grows Sales 261% on Instacart

Agenda

- Challenges of Building Business Metrics
- Importance of Metrics Stores
- The Power of Incremental Processing
- Testing and Monitoring
- Case Study
- Q&A

Challenges

Consistency	Scalability	Reliability
• Inconsistency in the metrics definition • Inconsistent metrics derived from different sources • Error introduced during clone and edit • Inconsistent application and enforcement of policies, such as PII, financial controls and cost-effectiveness.	• Batch processing with static lookback windows doesn’t scale well to increasing data volumes • Redundant reprocessing leads to a waste of time and computational resources • Slow processing delays the availability of insights for decision making	• Gaps in review • Insufficient testing • No unit test during development phase

Challenges

Consistency

- Inconsistency in the metrics definition
- Inconsistent metrics derived from different sources
- Error introduced during clone and edit
- Inconsistent application and enforcement of policies, such as PII, financial controls and cost-effectiveness.

Scalability

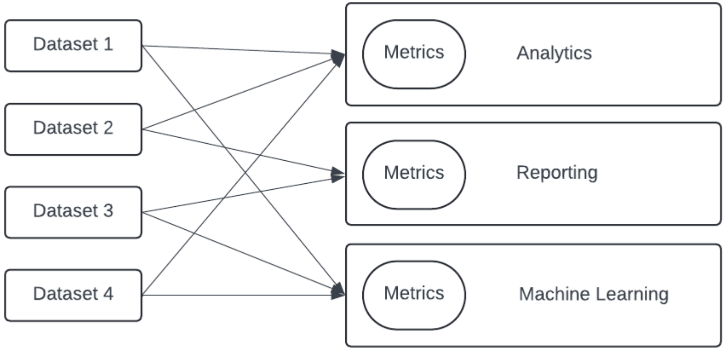
- Batch query doesn’t scale well to increasing data volumes
- Redundant reprocessing leads to a waste of time and computational resources
- Slow processing delays the availability of insights for decision making

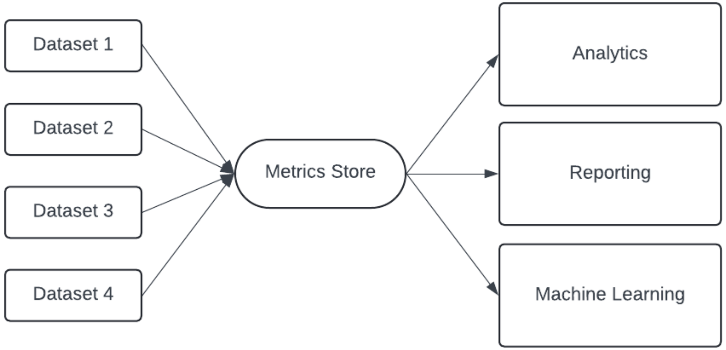
Reliability

- Gaps in review
- Insufficient testing
- No unit test during development phase

Metrics Store

- Centralized storage system for metrics
- Single source of truth for definition and data
- Data reusable across teams and applications
- Optimized for efficient computation and low-latency retrieval





Challenges

Consistency

- Inconsistency in the metrics definition
- Inconsistent metrics derived from different sources
- Error introduced during clone and edit
- Inconsistent application and enforcement of policies, such as PII, financial controls and cost-effectiveness.

Scalability

- Batch query doesn't scale well to increasing data volumes
- Redundant reprocessing leads to a waste of time and computational resources
- Slow processing delays the availability of insights for decision making

Reliability

- Gaps in review
- Insufficient testing
- No unit test during development phase

Incremental Processing

Batch Processing

- Accumulated large volume data
- Moderate to high latency
- Low complexity

Realtime Processing

- Data as it arrives
- Very low latency
- High complexity

Incremental Processing

- Only new/changed data
- Low to moderate latency
- Moderate complexity

Minimize Data Reprocessing

- Efficiency
    - Faster processing speed
    - Lower Infra cost
- Scalability
    - Load grows with the data change rate, not the total volume
    - Facilitate complex metrics calculation like cumulative metrics

Implementation Strategies with DBX

Structured Streaming + Checkpoint

- Use structured streaming and checkpoint to allow exact once processing
- No extra effort required to deal with late arrival data
- Example: Flatten JSON files into structured table

Change Data Feed

- Only process changed data in a stateless job
- Minimize the reprocessing window in a stateful job by identifying the earliest change data
- Example: Populate dimension table from event stream
- Example: Minimize budget consolidation reprocessing window

Challenges

Consistency

- Inconsistency in the metrics definition
- Inconsistent metrics derived from different sources
- Error introduced during clone and edit
- Inconsistent application and enforcement of policies, such as PII, financial controls and cost-effectiveness.

Scalability

- Batch query doesn’t scale well to increasing data volumes
- Redundant reprocessing leads to a waste of time and computational resources
- Slow processing delays the availability of insights for decision making

Reliability

- Gaps in review
- Insufficient testing
- No unit test during development phase

Testing And Monitoring

Code review, **Testing** and **Monitoring** are mandatory for Datasets maintained in the Metrics Store.

Each iterations made to the underlying pipelines is safeguarded by:

- **Unit testing:** Mock inputs and assert each component.
- **Monitoring:** Automatically generate data monitors for our pipelines.

Testing and Monitoring

Sample Scala Code for Unit Test

Spark Scala

```
test("Test metrics happy path") {
   val inputDfMap = mockedInputDfMap
   val expectedDf = generateDataFrame(schema, mockdata)
   val actualDf = Transform.apply(inputDfMap, configArgs, "testMetricName")
   assertDataFrameEquals(
     expectedDf,
     actualDf,
     strictColOrder = false,
     ignoreNullable = true
   )
}
```

Testing and Monitoring

Sample SQL Code for Data Quality Monitoring

SQL	JSON

`# Template to generate duplicate checks`
`select {{primary_key}}, count(*) as ct`
 	
`from {{table_name}}`
 	
`where true`
 	
`and event_date_time_utc >= current_timestamp - interval {{look_back_hour}} hour`
 	
`group by {{primary_key}}`
 	
`having ct > 1`
 	
`limit 10`
	`# Data check configuration`
	`[`
	`{`
	`"table_name": "table_name_example_1",`
	`"pipeline_type": "type_1",`
	`"priority": "P2",`
	`"primary_key": "event_id",`
	`"look_back_hour": 24`
	`},`
	`{`
	`"table_name": "table_name_example_2",`
	`"pipeline_type": "type_2",`
	`"priority": "P3",`
	`"primary_key": "event_id",`
	`"look_back_hour": 24`
	`}`
	`]`

Challenges

Consistency

- Inconsistency in the metrics definition
- Inconsistent metrics derived from different sources
- Error introduced during clone and edit
- Inconsistent application and enforcement of policies, such as PII, financial controls and cost-effectiveness.

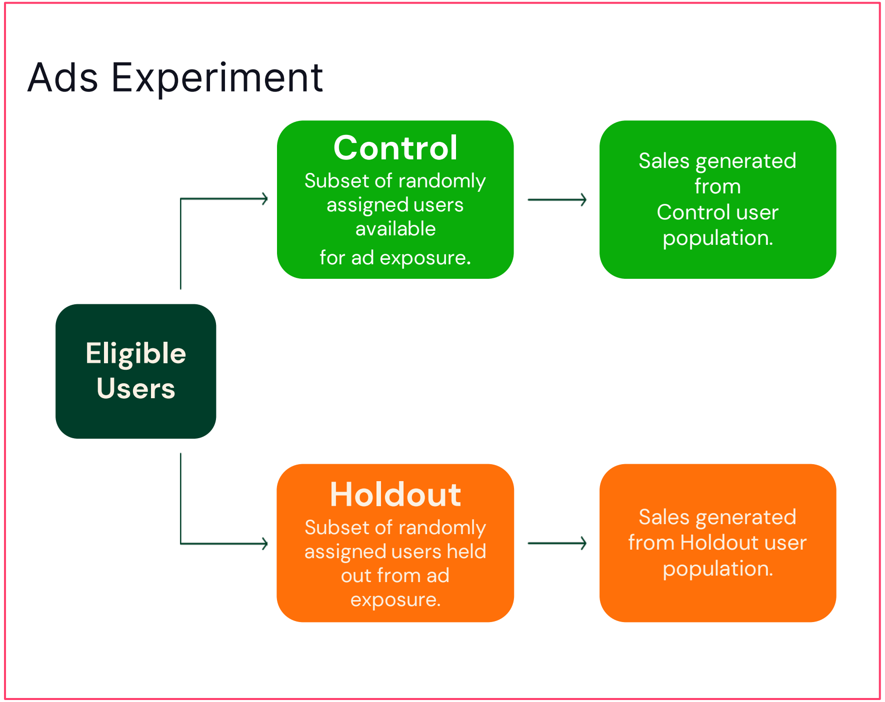
Scalability

- Batch query doesn't scale well to increasing data volumes
- Redundant reprocessing leads to a waste of time and computational resources
- Slow processing delays the availability of insights for decision making

Reliability

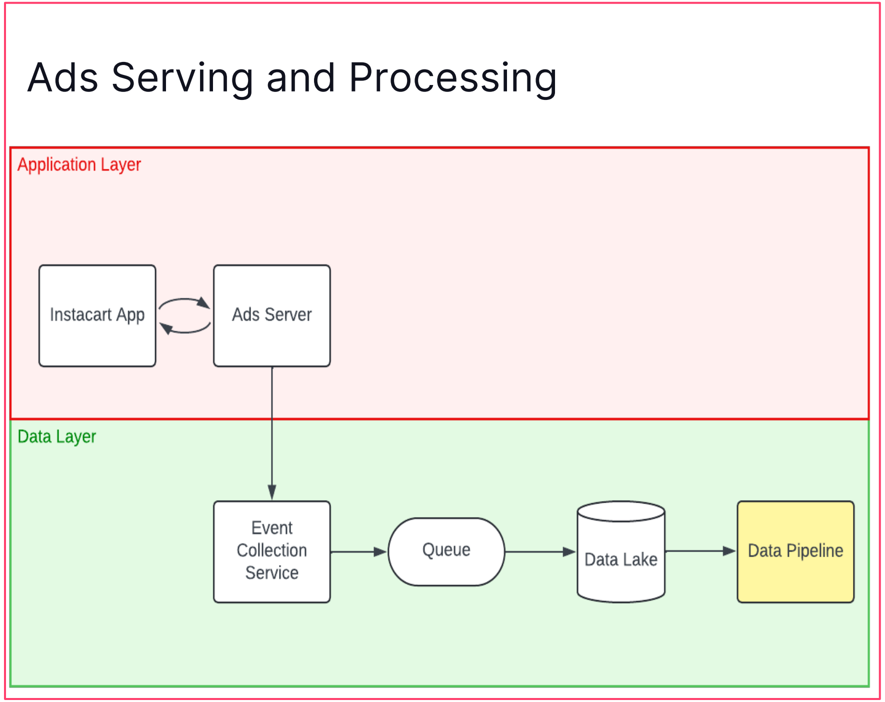
- Gaps in review
- Insufficient testing
- No unit test during development phase

Case Study - User Eligibility



- Build a user eligibility table for experimentation analysis
- User eligibility table stores the timestamp indicating when a user begins participating in each experiment

Case Study - User Eligibility



- Input - event stream emitted during assignment
- Output - dimension table of user eligibility with first assigned timestamp
- Metrics shared by monitoring, analysis, reporting
- *Structured Streaming, Checkpoint, Merge*

User Eligibility Calculation

Code Snippet for Batch Solution

SQL

```
SELECT
    experiment_id,
    experiment_type,
    variant,
    user_id,
    MIN(timestamp) AS first_assign_date_time_pt
FROM
    {event_table_name}
WHERE
    timestamp >= {experiment_start_timestamp}
GROUP BY 1,2,3,4
```

User Eligibility Calculation

Code Snippet for Structured Streaming Read

Spark Scala

```scala
def readStreamTimestamp(deltaPath: String,startingTimestamp: String)
   (
     implicit sparkSession: SparkSession,
   ): DataFrame = {
   log(s"Reading Stream from timestamp $startingTimestamp and path $deltaPath")
   sparkSession.readStream
     .format("delta")
     .option("startingTimestamp", startingTimestamp)
     .option("ignoreDeletes", "true")
     .option("ignoreChanges", "true")
     .load(deltaPath)
}
```

User Eligibility Calculation

Code Snippet for Merge Write

Spark Scala

```scala
deltaTableUserEligibility
  .alias("existing")
  .merge(
      dataFrame.alias("newData"),
      s"newData.USER_ID = existing.USER_ID" +
        s" AND newData.EXPERIMENT_ID = existing.EXPERIMENT_ID" +
        s" AND newData.EXPERIMENT_TYPE = existing.EXPERIMENT_TYPE" +
        s" AND newData.VARIANT = existing.VARIANT",
  )
```

User Eligibility Calculation

Code Snippet for Merge Write

Spark Scala

```
.whenNotMatched()
.insertExpr(
    Map(
        USER_ID -> s"newData.USER_ID",
        EXPERIMENT_ID -> s"newData.EXPERIMENT_ID",
        EXPERIMENT_TYPE -> s"newData.EXPERIMENT_TYPE",
        VARIANT -> s"newData.VARIANT",
        FIRST_ASSIGN_DATE_TIME_PT -> s"newData.FIRST_ASSIGN_DATE_TIME_PT",
        FIRST_ASSIGN_DATE_PT -> s"newData.FIRST_ASSIGN_DATE_PT",
    ),
)
```

User Eligibility Calculation

Code Snippet for Merge Write

Spark Scala

```scala
.whenMatched(s"newData.FIRST_ASSIGN_DATE_TIME_PT < existing.FIRST_ASSIGN_DATE_TIME_PT")
.updateExpr(
    Map(
        FIRST_ASSIGN_DATE_TIME_PT -> s"newData.FIRST_ASSIGN_DATE_TIME_PT",
        FIRST_ASSIGN_DATE_PT -> s"newData.FIRST_ASSIGN_DATE_PT",
    ),
)
```

Thank you!