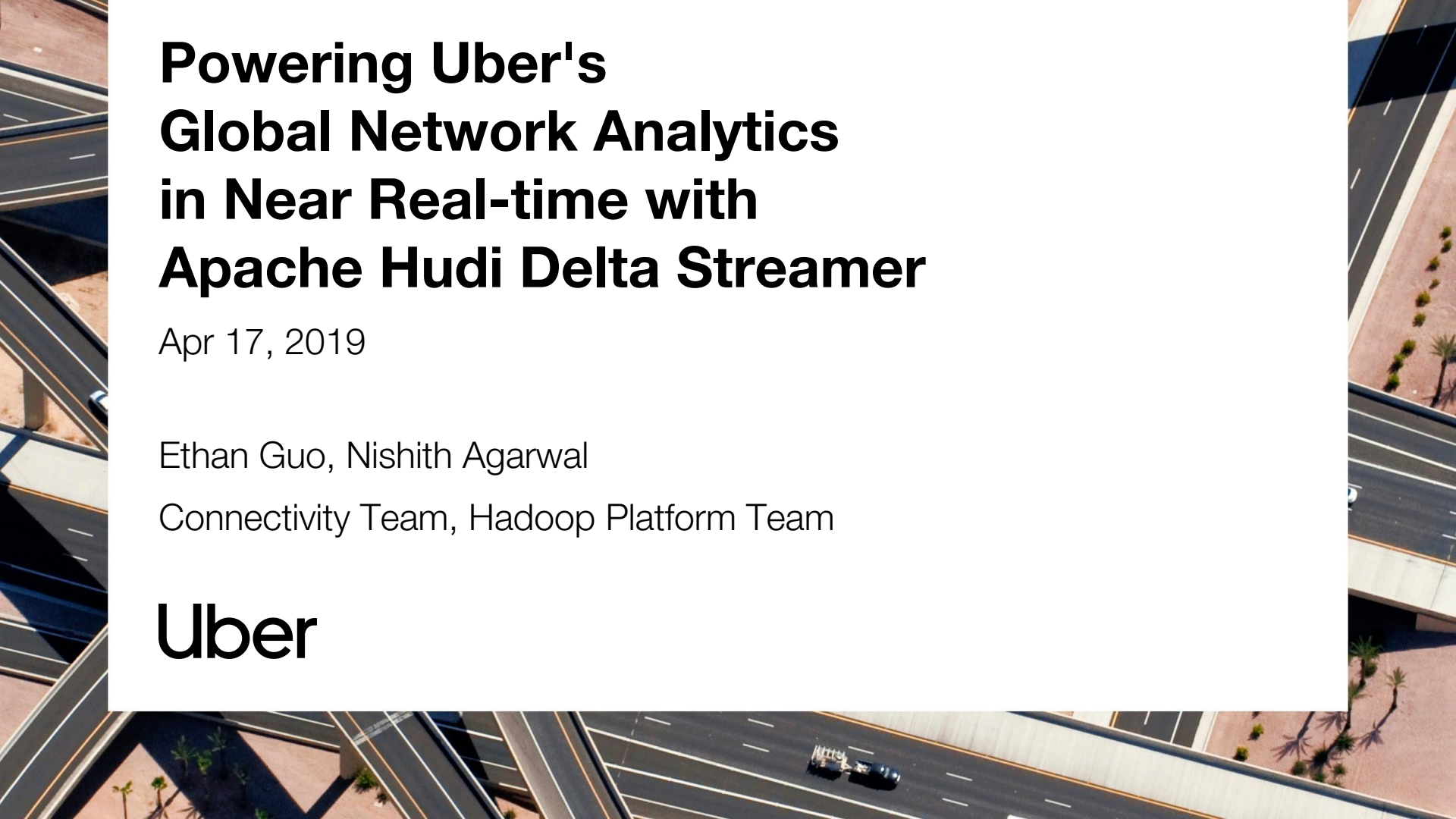


Powering Uber's Global Network Analytics in Near Real-time with Apache Hudi Delta Streamer

Apr 17, 2019

Ethan Guo, Nishith Agarwal

Connectivity Team, Hadoop Platform Team

Uber

Agenda

01 Motivation
02 Apache Hudi Incremental Pulls
03 Building Incremental Pipelines
04 Experience @ Uber

Motivation

Scale, Problem

Transportation at Scale

We ignite opportunity by setting the world in motion.

600+
Cities

64
Countries

6
Continents

10B
Cumulative trips

Network Monitoring

Heavy mobile network usage

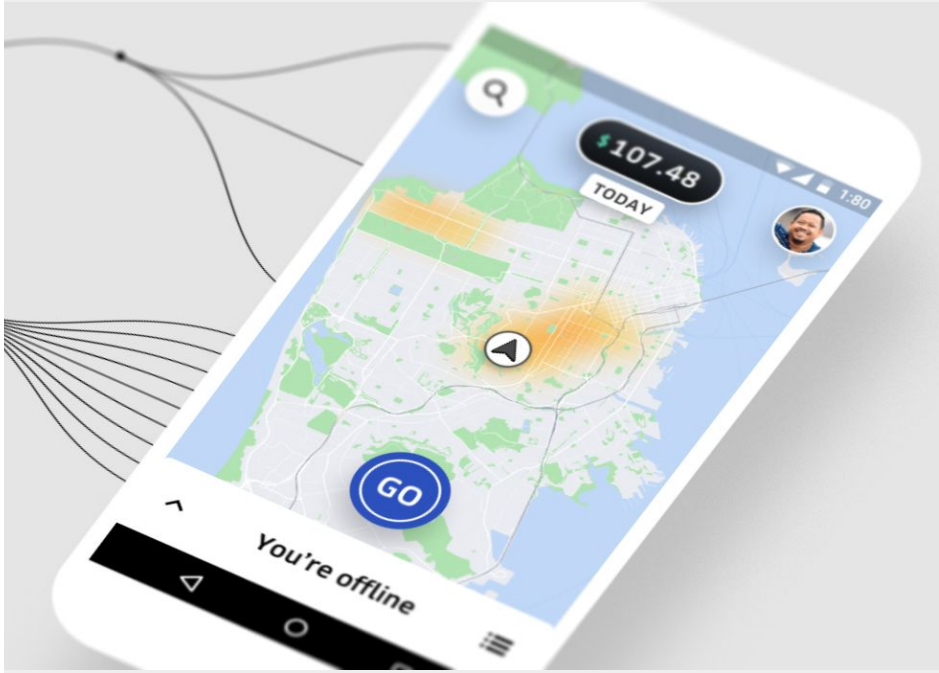
Mobile first: services through apps

- Riders, drivers, eats, new mobility (bikes, scooters), etc.

Apps rely 100% on wireless networks

- **70%** of requests in rider app use cellular (LTE/3G/2G)
- **92%** of requests in driver app use cellular

Near real-time monitoring of network reliability/perf required!



Performance - High Dimensionality

Performance of wireless networks highly varies across the globe

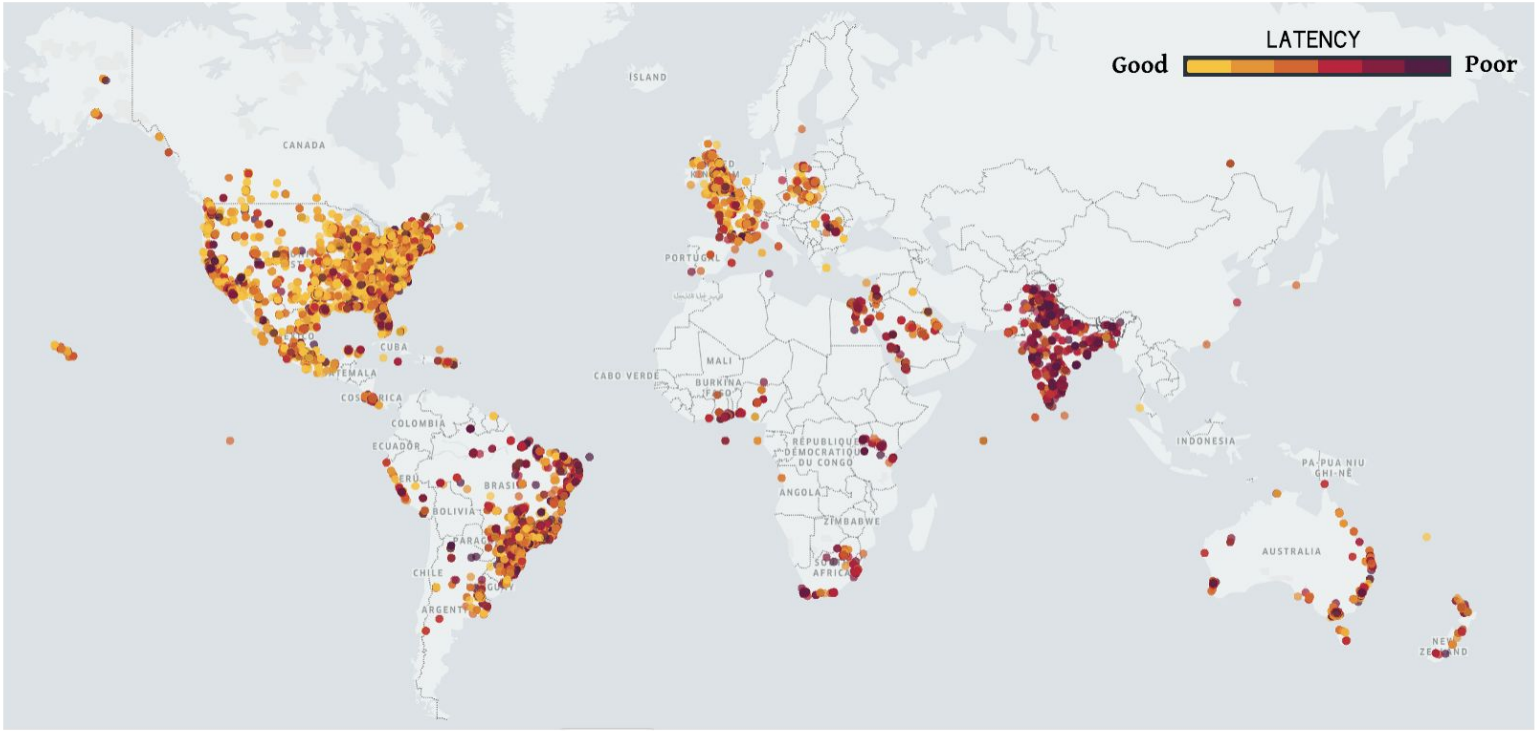


Fig: Tail-end network latencies across the major cities where Uber operates

Performance - High Dimensionality

Other dimensions

Time

- Load variance, congestion

ISP network quality

- Cellular carriers

Network connectivity / coverage

- LTE vs 3G vs EDGE/2G vs WiFi

Protocol

- HTTP/1.1 vs H2 vs H3

Domain names

User mobility

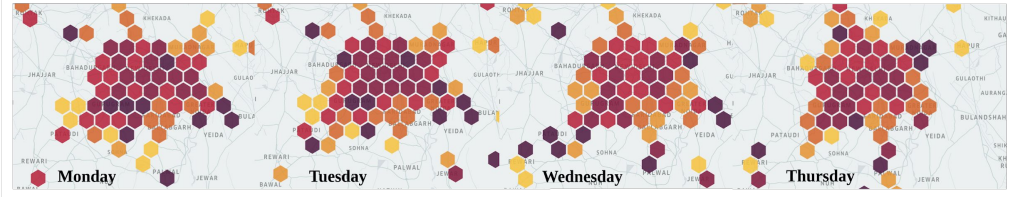


Fig 1: Tail-end network latencies for 2km hex across different days of week in Delhi, India

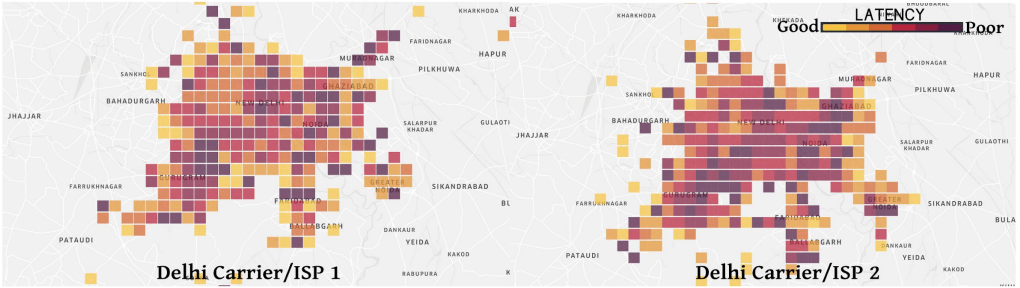


Fig 2: Tail-end network latencies for 2km hex across two major carriers in Delhi, India

Visibility into Network Performance

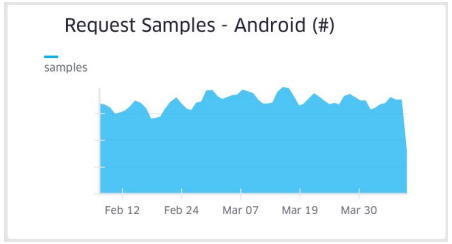
Multiple KPIs with dimension breakdowns

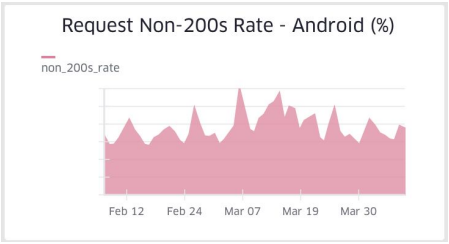
Challenges

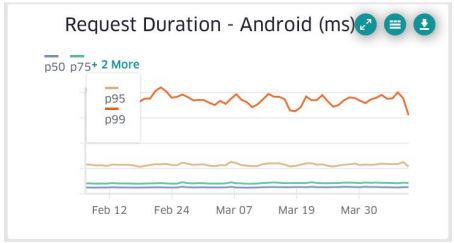
- High dimensionality, on the order of 10M
- Large data volume: subject to sampling, only last few days in ELK
- Custom filters and calculation

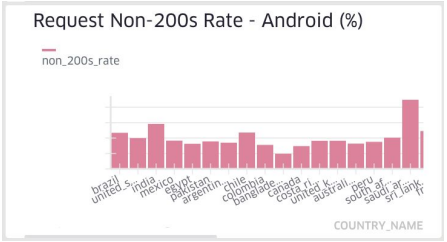
Customized Performance Dashboard

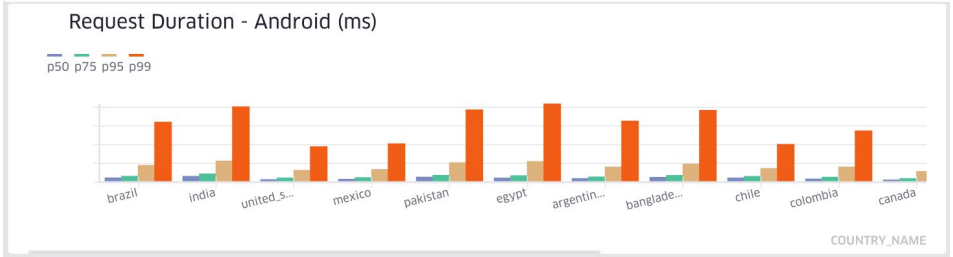
- Apache Spark jobs generating perf summary in Apache Hive tables, with months of history
- Latency and error rate metrics across several dimensions
- Hours -> minutes with a few clicks by on-call engineers





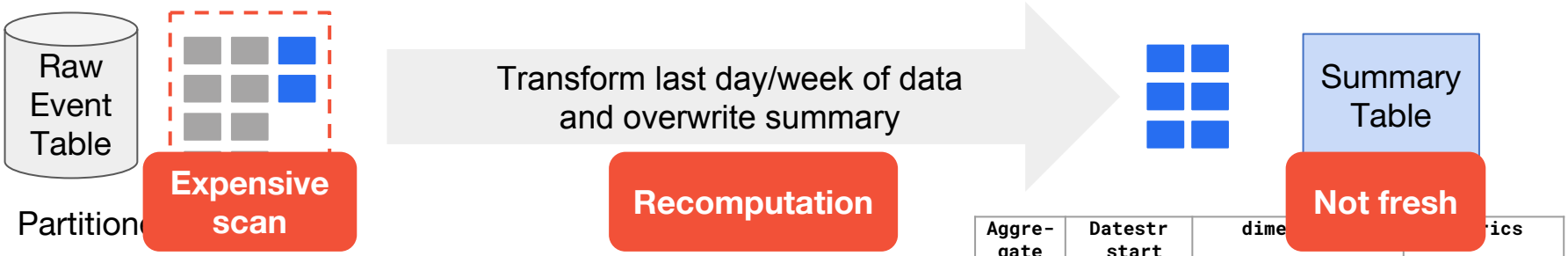






Batch Job Updating Metrics (Legacy)

Scalability challenges and ineffective use of resources



Aggre-gate	Datestr _start	dime	rics
day	2019-04-01	{“country”: “united_states”}	{“request_duration_p50”: 100.0}
day	2019-04-01	{“country”: “united_states”, “network_type”: “lte”}	{“non_200s_rate”: 0.5}
week	2019-04-01	{“protocol”: “h2”}	{“error_samples”: 10}

(Simplified schema for demonstration)

Scalability issues

- Expensive table scan/read: repeated reading of data
- Recomputation on same data: inefficient use of resources
- Summary not fresh: long time to refresh the metrics (3h~5h)

Incrementals to the Rescue

Incrementally pulls new data and updates performance metrics

gate	_start	dimensions	metrics
day	2019-04-01	{“country”: “united_states”}	{“request_duration_p50”: 100.0}
day	2019-04-01	{“country”: “united_states”, “network_type”: “lte”}	{“non_200s_rate”: 0.5}
week	2019-04-01	{“protocol”: “h2”}	{“error_samples”: 10}

(Simplified schema for demonstration)

A more efficient way

- Scan and process only new/updated entries in data source
- Update existing performance metrics with incrementals

Apache Hudi Incremental Pulls

Model for efficient processing

Apache Hudi

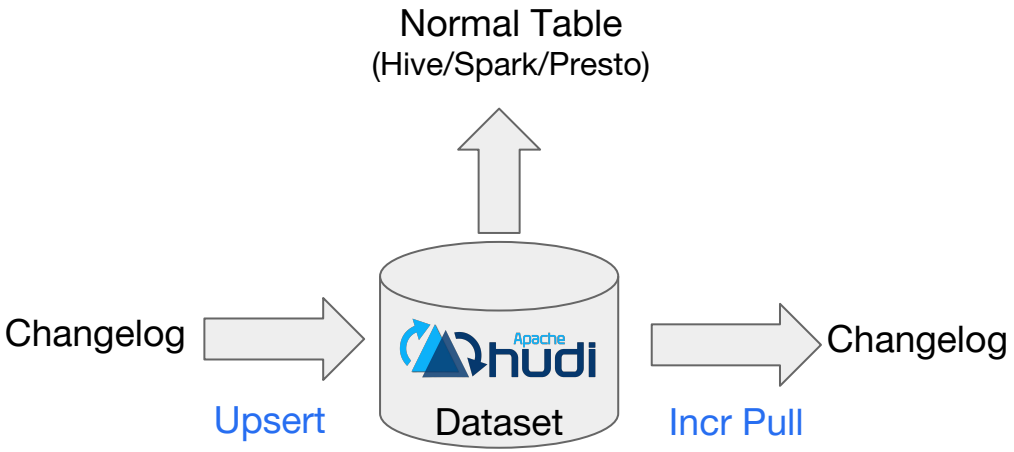
Stream style processing on big data

Turn batch jobs to incremental model

- Improve latency by incorporating only deltas
- Scale more by avoiding recomputation, lowering cost

Spark Library for

- Mutations to datasets
- Changelogs from datasets
- Manage files efficiently
- Provide snapshot isolation
- Upsert, insert, incremental pull primitives



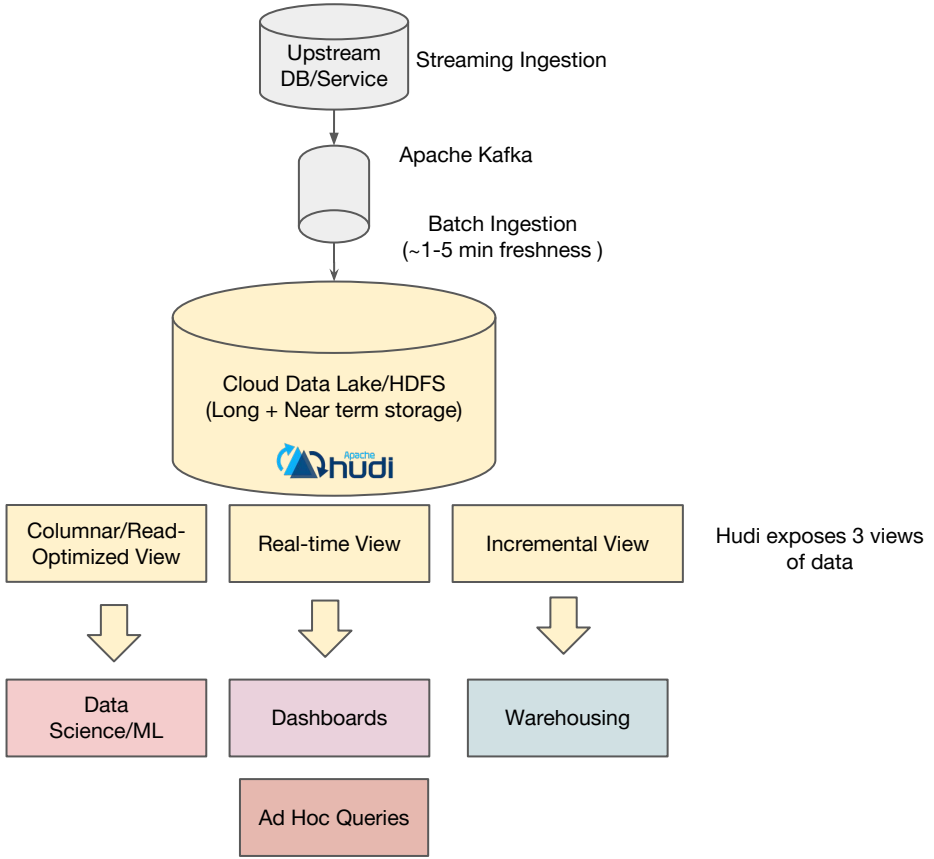
Open Source

- https://github.com/apache/incubator-hudi
- https://hudi.apache.org/

Hudi-Based Architecture

Unified analytical storage

- Complete file management on a data lake, including features such as file count, file sizing, data layout and more.
- Exposes different views of data tailored for use-cases!
- Full access to the organization's data across variety of needs!
- Greatly reduces operational footprint, specialized DBs only for special needs.

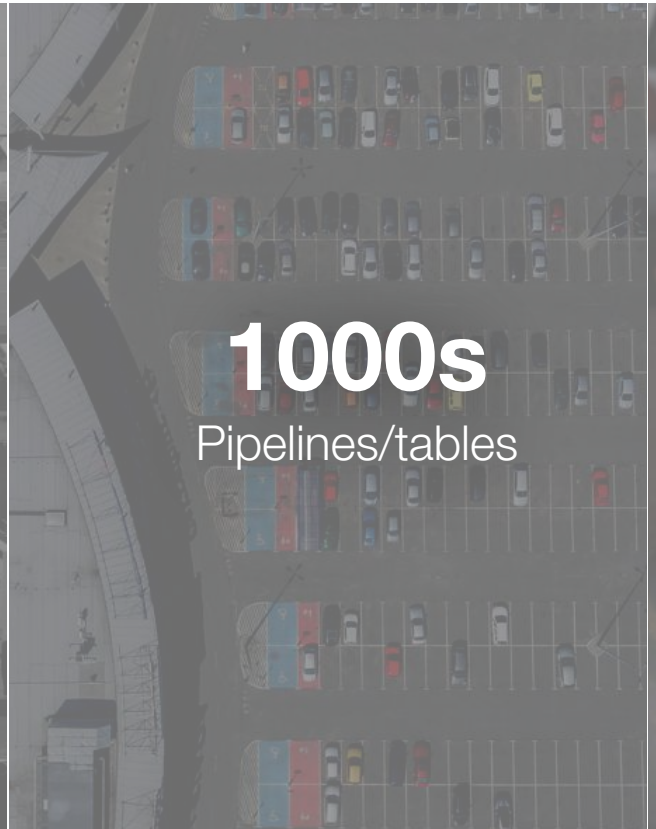


Hudi @ Uber

The facts and the figures

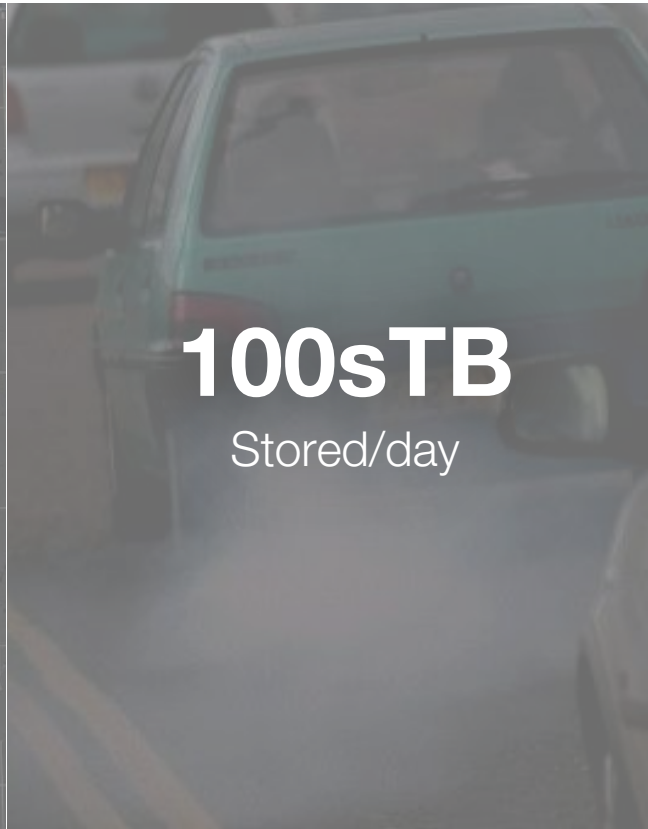
10s PB

Entire Hadoop Data Lake



1000s

Pipelines/tables



100sTB

Stored/day

Incremental Model

Stream style processing on batch data

Near real-time results

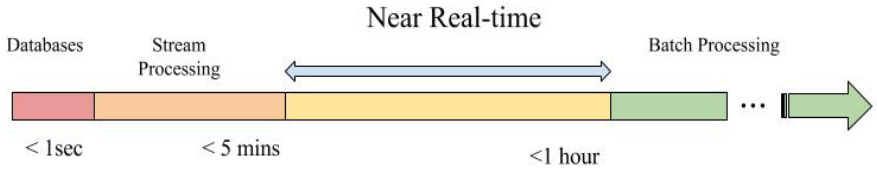
- Mini batch jobs, every few minutes

Upsert (Primitive #1)

- Modify processed results
- Like state stores in stream processing

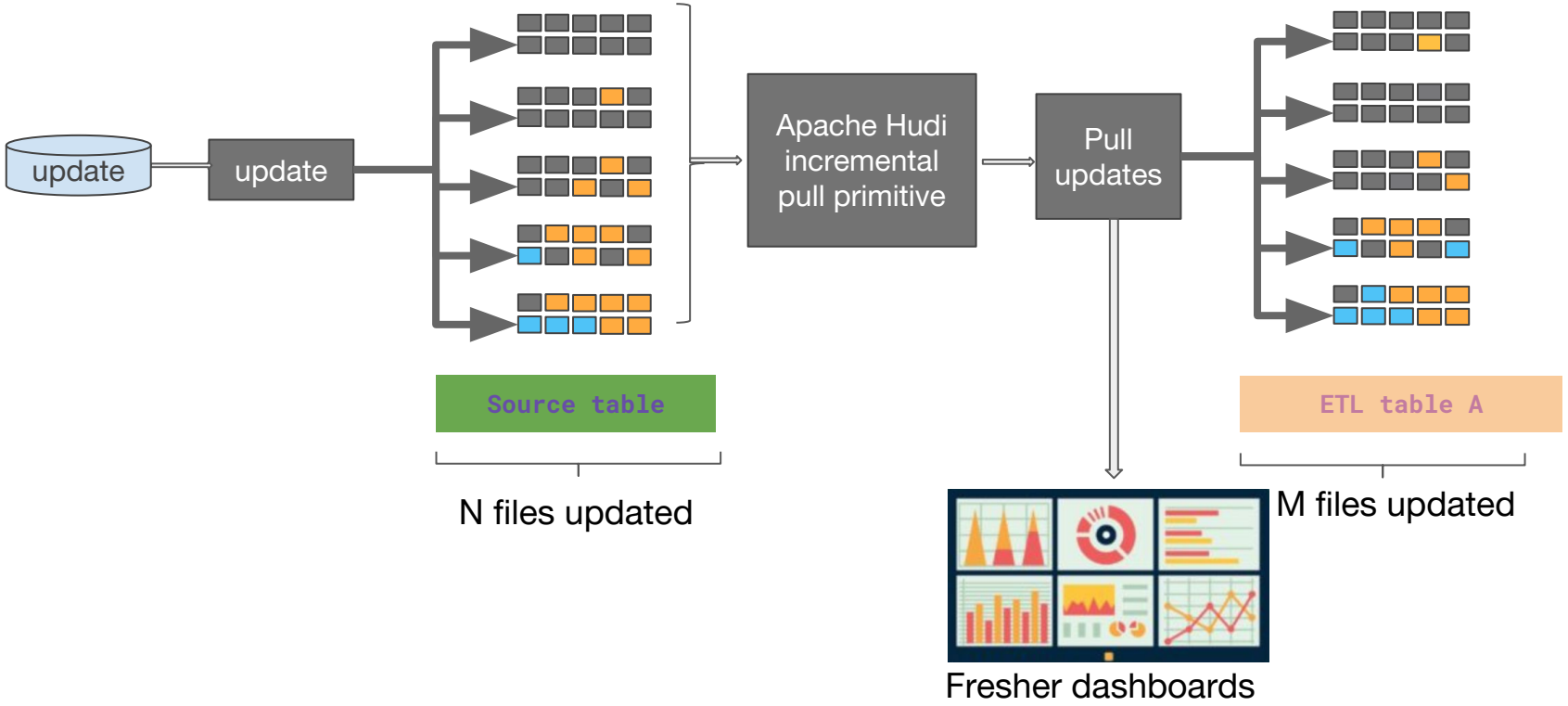
Incremental Pull (Primitive #2)

- Log stream of changes, avoids costly scans
- Faster flow of data to next stage in dataflow



Incremental Pipelines (ETL) and Dashboarding

Incremental processing



Using Apache Hudi

Popular ways to manage Apache Hudi datasets

- **Spark DataSource API** to help read/write Hudi datasets

```
IncrementalPull -> Dataset<Row> hoodieIncViewDF =
spark.read().format("com.uber.hoodie")...

Upsert -> inputDataset.write.format("com.uber.hoodie")...
```

- **HudiDeltaStreamer**, an end-to-end ingestion framework with configurable sources, schema repositories and built-in support for using Hudi to read/write datasets

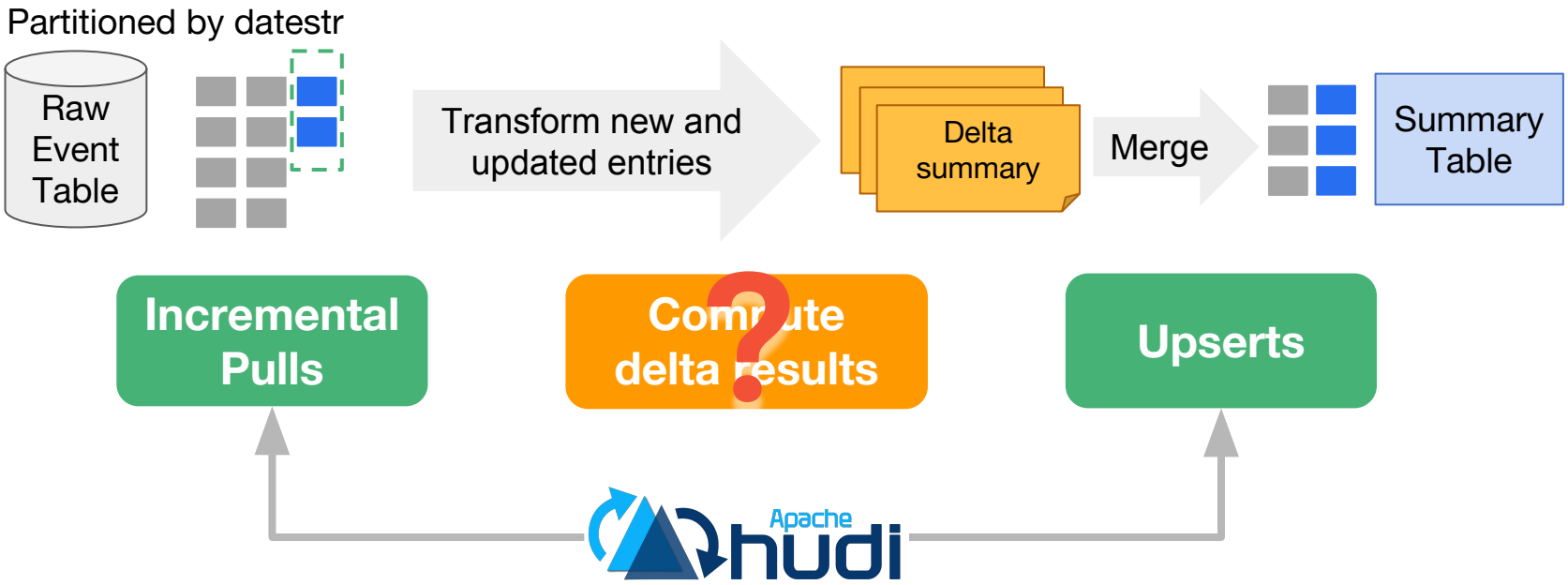
```
IncrementalPull & Upsert -> spark-submit --class
com.uber.hoodie.utilities.deltastreamer.HoodieDeltaStreamer
--source HudiIncrementalSource --schemaProvider .. --operation
UPSERT …
```

Building Incremental Pipelines

Update network metrics incrementally

Incremental Pipeline

Components to update metrics incrementally



Update Metrics Incrementally

Sketching of incremental data

Categories of metrics

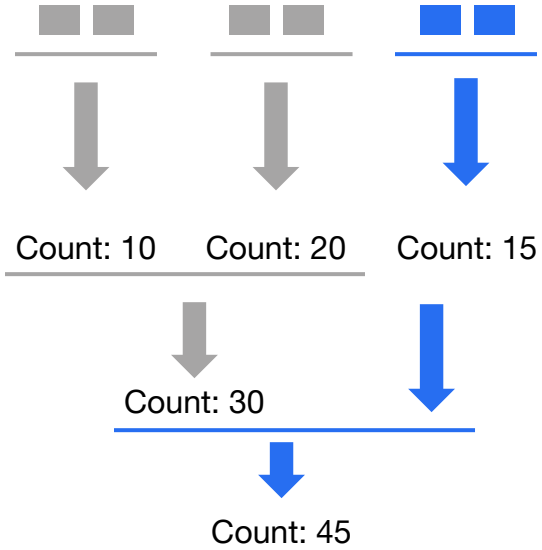
- Count: network calls, error samples
- Ratio/Percentage: HTTP non-200s rate, % traffic on H2
- Percentiles: p50, p95, p99 of request duration

Requirements

- Metric update only depends on incrementals and intermediate results if necessary
- Intermediate results much smaller than raw events

Sketching - Generate Mergeable Summary

- Small data structure to calculate or approximate metrics
- Appropriate algorithms for all categories of network metrics



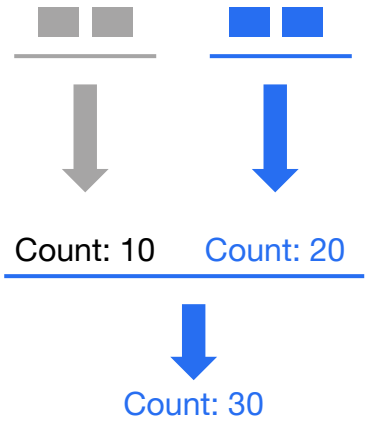
Sketching for Count and Ratio Metrics

Counting of samples

Count

(Request, error samples)

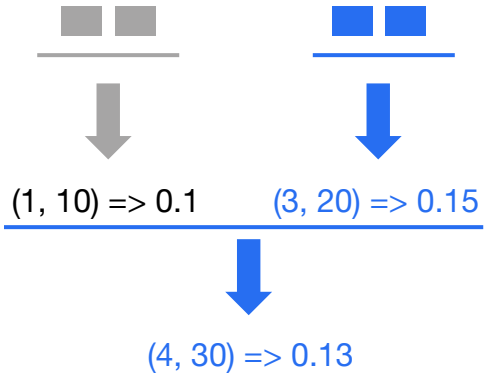
- Count = # of samples in incrementals
- Sum of count => total count



Ratio / Percentage

(HTTP non-200s rate, % traffic on H2)

- Ratio = num. / denom., storing counts of numerator and denominator from incrementals
- (Sum num.) / (Sum denom.) => overall ratio



Sketching for Percentile Metrics

Using t-Digests

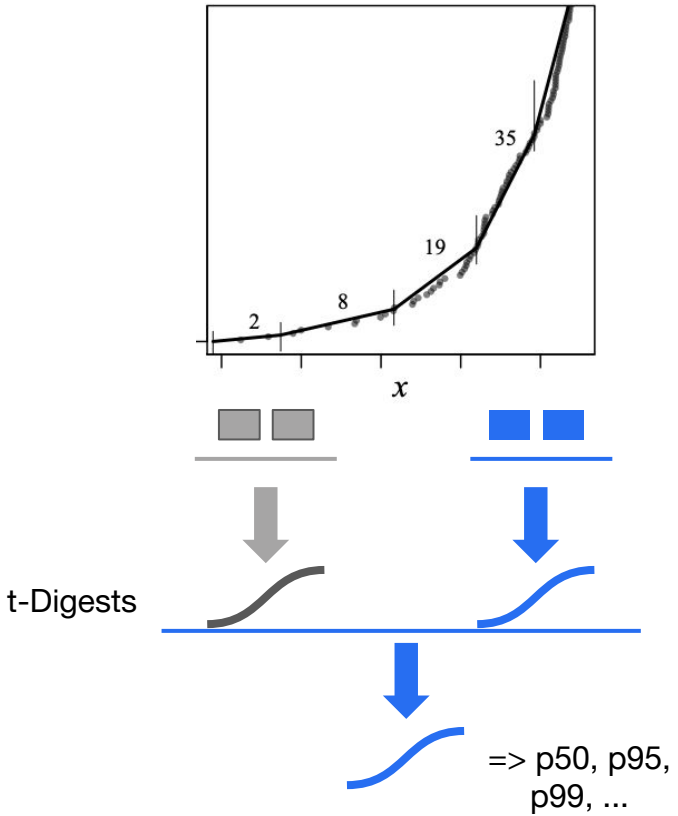
Percentiles

(p50, p95, p99 of request duration)

- Need sketching of the distributions
- Use t-Digest

What is a t-Digest?

- A data structure formed by clustering real-valued samples with variable-sized bins [1]
- Give high accuracy near the tails of a distribution with small sketches
- No loss in accuracy when combining t-Digests from multiple skewed distributions
- Knob to tune tradeoff between accuracy and storage footprint



[1] Computing Extremely Accurate Quantiles Using t-Digests, Ted Dunning, Otmar Ertl, https://raw.githubusercontent.com/tdunning/t-digest/master/docs/t-digest-paper/histo.pdf

Storing Sketches

Intermediate Apache Hive table

Sketches in a separate Apache Hive table

- Sketches kept at daily level for all dimension breakdowns
- Incrementals of raw events update sketches
- Serialized t-Digest stored in the table

Metrics transformed from sketches

- New/updated sketches update performance summary metrics under same dimensions

Raw network events

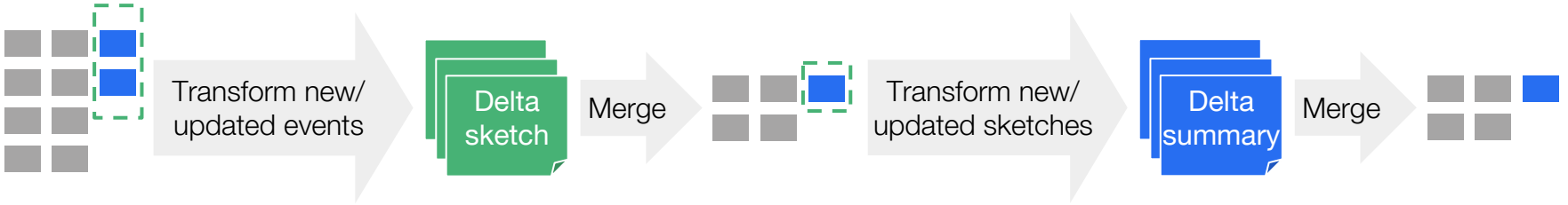
Aggregate	Datestr_start	dimensions	metrics_category	double_sketches	tdigest
day	2019-04-01	{“country”: “united_states”}	request_duration	{“total_samples”: 1000.0}	{“request_duration”: “...”}
day	2019-04-01	{“country”: “united_states”, “network_type”: “lte”}	non_200s_rate	{“non_200s_samples”: 5.0, “total_samples”: 600.0}	null

Aggregate	Datestr_start	dimensions	metrics
day	2019-04-01	{“country”: “united_states”}	{“request_duration_p50”: 100.0}
day	2019-04-01	{“country”: “united_states”, “network_type”: “lte”}	{“non_200s_rate”: 0.83}

(Simplified schema for demonstration)

Pipeline Overview

Two-stage incremental updates



Aggre-gate	Datestr_ start	dimensions	Metrics_ category	Double_ sketches	tdigest
day	2019-04 -01	{"country": "united_states"}	request_durat ion	{"total_ samples": 1000.0}	{"request_ duration": "..."}
day	2019-04 -01	{"country": "united_states", "network_type": "lte"}	non_200s_rate	{"non_200s_ samples": 5.0, "total_ samples": 600.0}	null

Aggre-gate	Datestr_ start	dimensions	metrics
day	2019-04-01	{"country": "united_states"}	{"request_ duration_p50": 100.0}
day	2019-04-01	{"country": "united_states", "network_type": "lte"}	{"non_200s_ rate": 0.5}

(Simplified schema for demonstration)

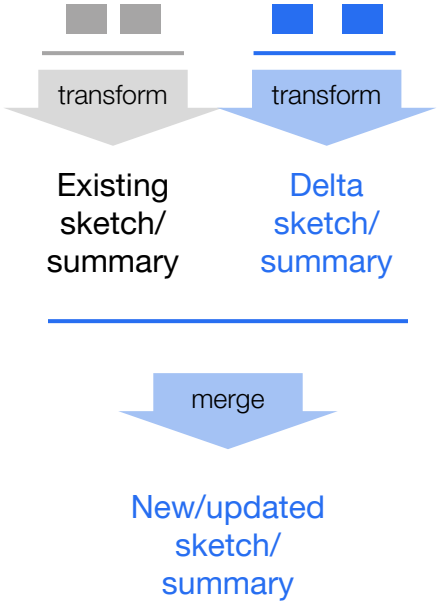
Implementation with Hudi Delta Streamer

Transforming incrementals with Hudi `Transformer` interface

- Pre-process and transform new/updated data into incremental sketch/summary

Updates sketches/summaries with `HoodieRecordPayload` interface

- Merge the sketches/summaries under same dimensions




```
class NetPerfSketchTransformer
  extends Transformer {
  def apply(
    jsc: JavaSparkContext,
    sparkSession: SparkSession,
    rowDataset: Dataset[Row],
    properties: TypedProperties
    ): Dataset[Row] = {
    ...
  }
}
```

```
class UpdateDataSketch(
  record: GenericRecord,
  orderingVal: Comparable[_])
  extends BaseAvroPayload(
    record, orderingVal)
  with HoodieRecordPayload[
    UpdateDataSketch] {
  ...

  @throws[IOException]
  override def
    combineAndGetUpdateValue(
      currentValue: IndexedRecord,
      schema: Schema
      ): Optional[IndexedRecord] = {
      ...
    }
  ...
}
```

Experience @ Uber

Testing, Deployment, & Monitoring

Production Incremental Pipeline Setup

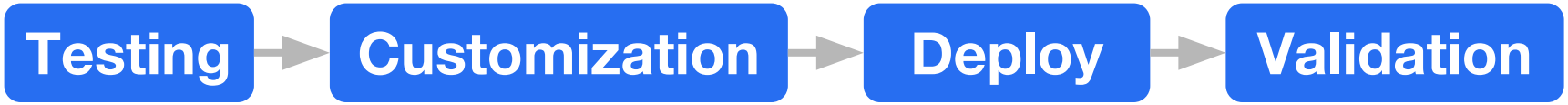
Testing jobs with spark-submit

- Apache Hudi configs stored in properties file and passed in to Delta Streamer class
- Out-of-box Apache Hudi metrics to grafana dashboard for monitoring
  - Total duration (transformation + commit), number of new files added, number of records added/updated, etc.

Customization

- Predicate pushdown and projection pruning (2-3x improvements on reading data)
- Fine tuning of shuffle partitions (`spark.sql.shuffle.partitions`)

Production Incremental Pipeline Setup



Deploy to Hadoop Yarn cluster using Apache Zeppelin

- Apache Zeppelin: Web-based notebook that enables data-driven, interactive data analytics, out-of-box support for Spark/Hadoop
- Setup crontab for incremental pipelines to run every 1 hour

Validate performance summary with batch pipelines

- Run incremental and batch jobs in parallel
- Compare performance metrics of dates with complete data between two pipelines
  - Count/radio metrics match
  - Percentile metrics with less than 0.5% error for large sample size (>10k)

Pipeline Runtime

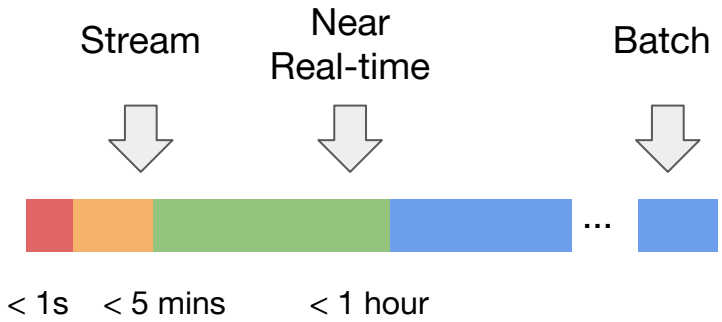
Efficient table scan/read, computation

Data size

- Billions of records and 100s GB per day

Batch update pipeline

- **1200 cores** = 240 executors * 5 cores each
- **3-4 h**: **10-20 min** on read, **>2.5 h** on compute/write



Incremental pipeline

- **150 cores** = 150 executors * 1 core each
- Sketching: **30~40 min**, **~6 min** on read, **~30 min** on compute/write. ~5GB/day of sketches.
- Summary: **<30 min**, **~2 min** on read, **~20 min** on compute/write. ~4GB/day of summary.

8x improvements on resources, 4x on freshness

We ARE HIRING!

Reach out to us

hadoop-platform-jobs@uber.com


Powering Uber's
Global Network Analytics
in Near Real-time with
Apache Hudi Delta Streamer
Apr 17, 2019
Q&A