Robust, Scalable, Real-Time Event Time Series Aggregation at Twitter

Peilin Yang, Srikanth Thiagarajan, Jimmy Lin

Data Infrastructure Engineering Team

Twitter, Inc.

#OUTLINE

1) The Challenges
2) How do we tackle the challenges?
3) Case Study: Tweets Engagement
4) Takeaways

The Challenges

#SCALE

~500 Million Tweets/day

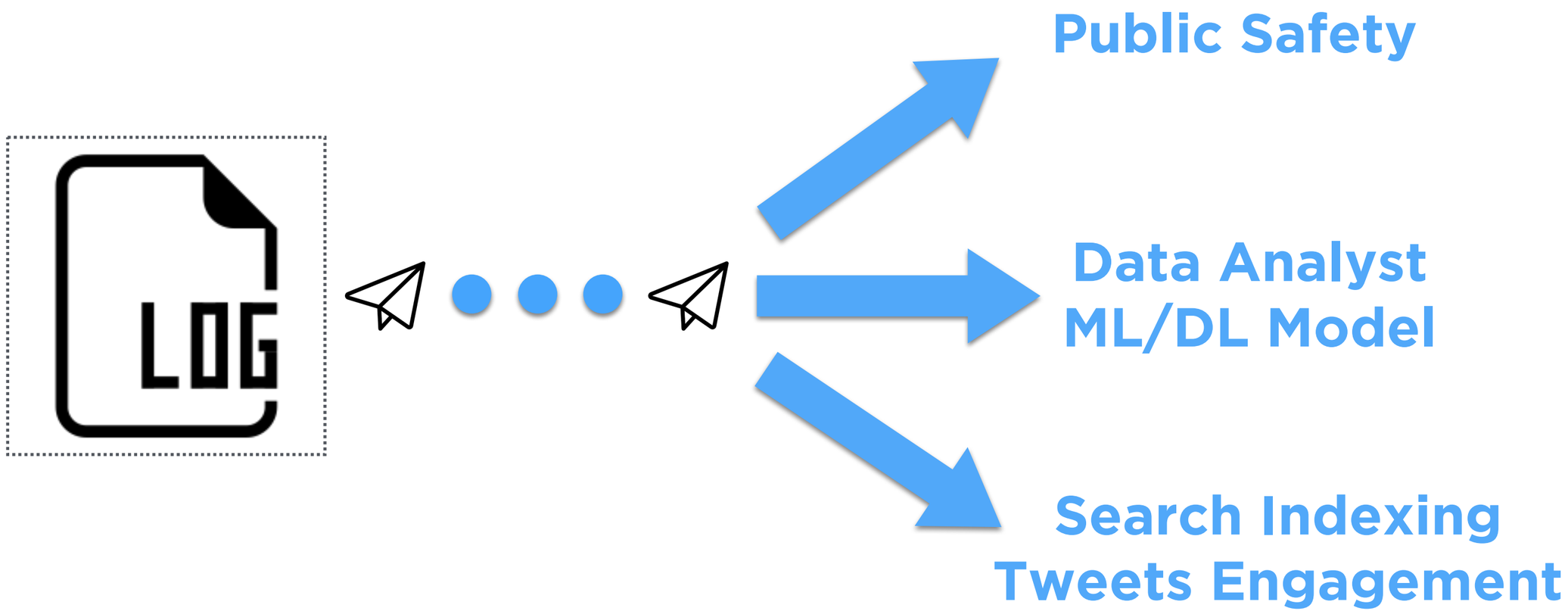
~5,000 Tweets/second

~350 Billion Events/day

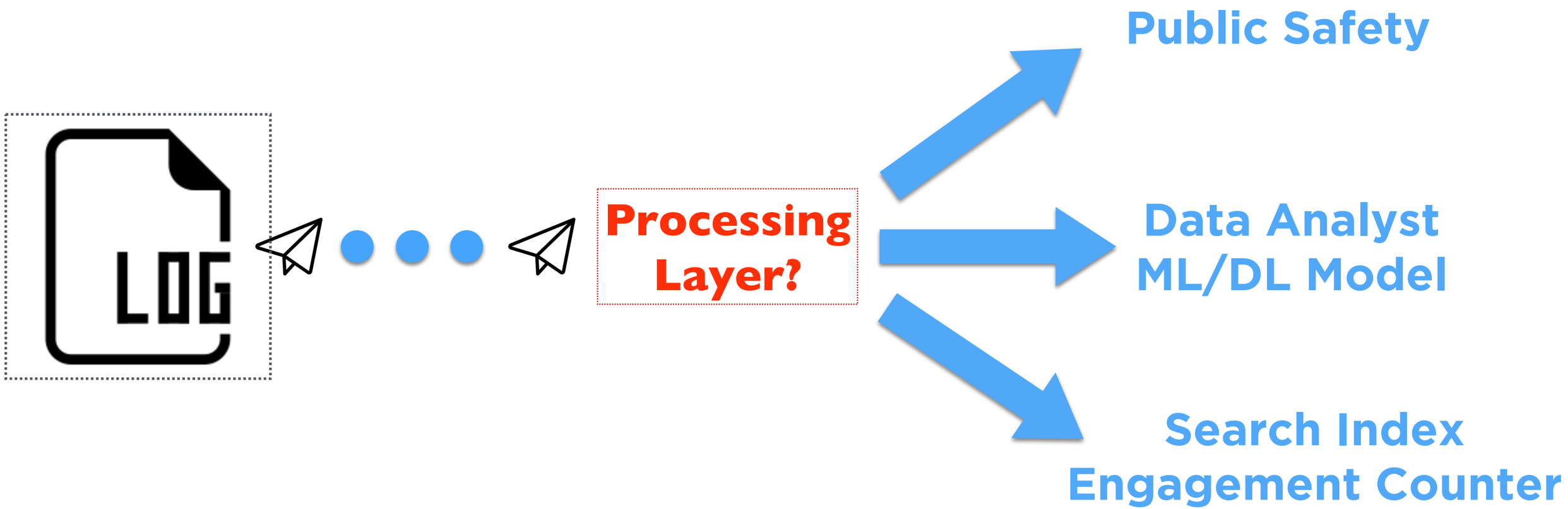
~4 Million Events/second

George Lee, Jimmy Lin, Chuang Liu, Andrew Lorek, and Dmitriy Ryaboy. The Unified Logging Infrastructure for Data Analytics at Twitter. *Proceedings of the VLDB Endowment*, 5(12):1771-1780, 2012.

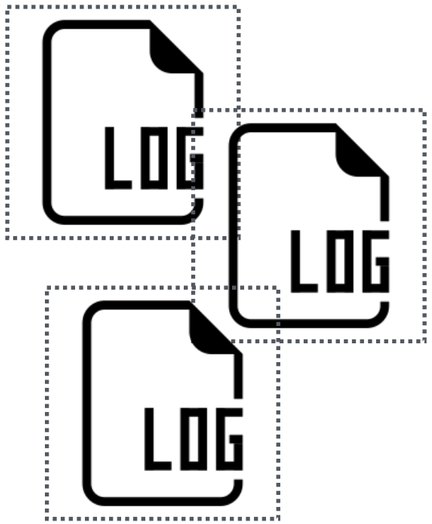
#REAL-TIME PROCESSING



#REAL-TIME PROCESSING



#REAL-TIME PROCESSING

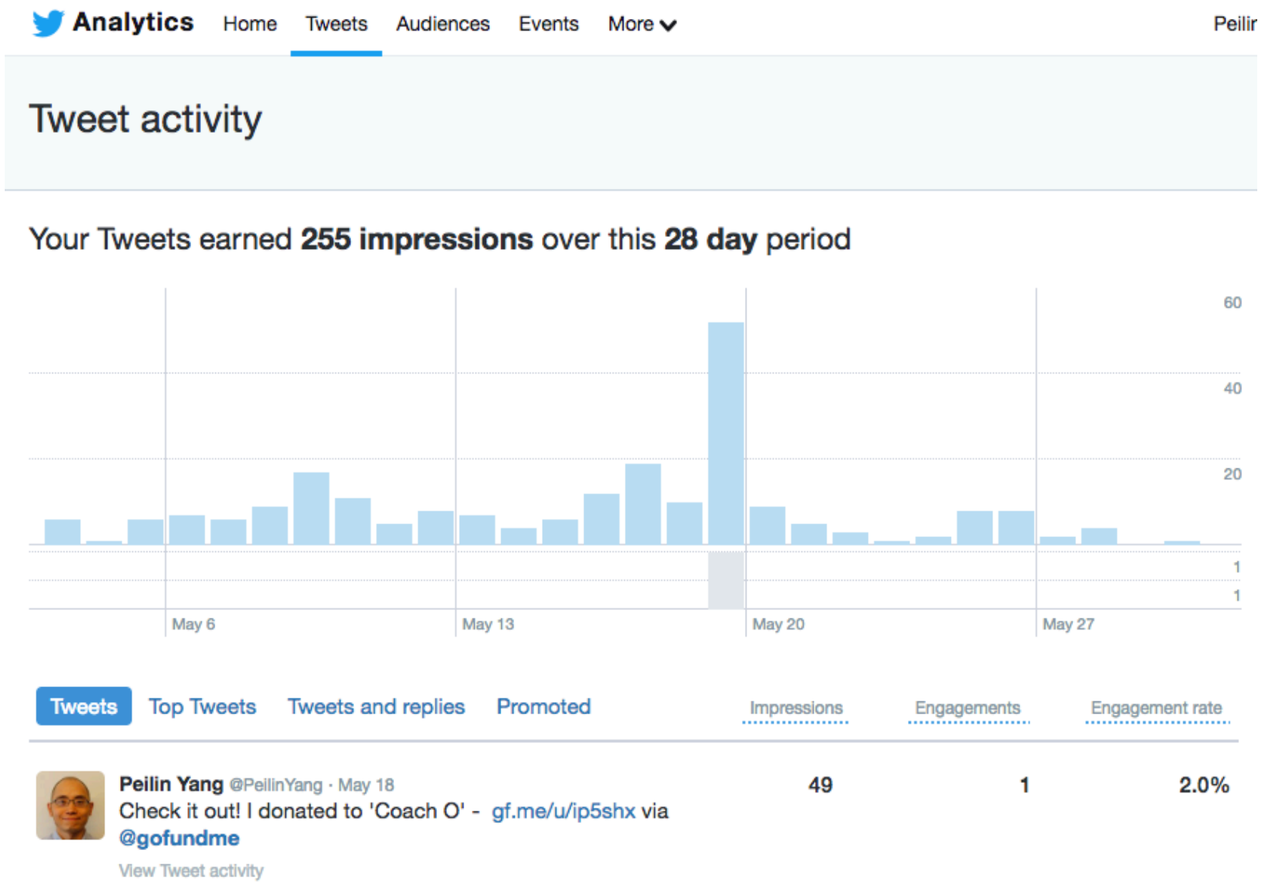


Processing → **Public Safety**

Processing → **Data Analyst**
ML/DL Model

Processing → **Search Index**
Engagement Counter

#EXAMPLE – TWEETS ENGAGEMENT



- Tweets Engagement shows how many engagements your tweets have received historically and bucketed in hours.
- Data is available after 10 seconds the tweet publishes. (**real-time**)
- Data will be validated after 24 hours for accurately charging the ads customers (**batch**).

Task is Defined as : Processing Twice (Batch + Real-time)

Who We Are

Data Infrastructure Engineering Team

We provide data processing solutions

#BATCH + REAL-TIME (PRE-2014)

- Pig (batch) + Storm (real-time)
- Later on Scalding (batch) + Storm (real-time)
- It was hard to maintain two sets of codes at the same time

#SUMMINGBIRD (2014)

- Declarative Streaming Map/Reduce DSL
- Real-time platform that runs on Storm
- Batch platform that runs on Hadoop
- Batch / Real-time Hybrid platform
- https://github.com/twitter/summingbird

Oscar Boykin, Sam Ritchie, Ian O'Connell, and Jimmy Lin. Summingbird: A Framework for Integrating Batch and Online MapReduce Computations. Proceedings of the VLDB Endowment, 7(13):1441-1451, 2014.

#SUMMINGBIRD

```
object OuroborosJob {
  def apply[P <: Platform[P]](source: Producer[P, ClientEvent], sink: P#Store[OuroborosKey, OuroborosValue]) =
    source.filter(filterEvents(_))
      .flatMap { event =>
      val widgetDetails = event.getWidget_details
      val referUrl: String = widgetDetails.getWidget_origin
      val timestamp: Long = event.getLog_base.getTimestamp
      val widgetFrameUrlOpt: Option[String] = Option(widgetDetails.getWidget_frame)
      for {
        tweetId: java.lang.Long <- javaToScalaSafe(event.getEvent_details.getItem_ids)
        timeBucketOption: Option[TimeBucket] <- timeBucketsForTimestamp(timestamp)
      } yield {
        val urlHllOption = canonicalUrl(referUrl).map(hllMonoid.create(_))
        val widgetFrameUrlsOption = widgetFrameUrlOpt map { widgetUrl: String =>
          widgetFrameUrlsSmMonoid.create((referUrl, (widgetFrameUrlSetSmMonoid.create((widgetUrl, 1L)), 1L)))
        }
        val impressionsValue: OuroborosValue = RawImpressions(
          impressions = 1L,
          approxUniqueUrls = urlHllOption,
          urlCounts = Some(embedCountSmMonoid.create((referUrl, 1L))),
          urlDates = Some(embedDateSmMonoid.create((referUrl, timestamp))),
          frameUrls = widgetFrameUrlsOption
        ).as[OuroborosValue]
        Seq(
          (OuroborosKey.ImpressionsKey(ImpressionsKey(tweetId.longValue, timeBucketOption)), impressionsValue),
          (OuroborosKey.TopTweetsKey(TopTweetsKey(timeBucketOption)), topTweetsValue)
        )
      }
    }.sumByKey(store)
      .set(MonoidIsCommutative(true))
}
```

Filter Events

Generate KV Pairs

Sum into Store

#SUMMINGBIRD

```
object OuroborosJob {
  def apply[P <: Platform[P]](source: Producer[P, ClientEvent], sink: P#Store[OuroborosKey, OuroborosValue]) =
    source.filter(filterEvents(_))
      .flatMap { event =>
      val widgetDetails = event.getWidget_details
      val referUrl: String = widgetDetails.getWidget_origin
      val timestamp: Long = event.getLog_base.getTimestamp
      val widgetFrameUrlOpt: Option[String] = Option(widgetDetails.getWidget_frame)
      for {
        tweetId: java.lang.Long <- javaToScalaSafe(event.getEvent_details.getItem_ids)
        timeBucketOption: Option[TimeBucket] <- timeBucketsForTimestamp(timestamp)
      } yield {
        val urlHllOption = canonicalUrl(referUrl).map(hllMonoid.create(_))
        val widgetFrameUrlsOption = widgetFrameUrlOpt map { widgetUrl: String =>
          widgetFrameUrlsSmMonoid.create((referUrl, (widgetFrameUrlSetSmMonoid.create((widgetUrl, 1L)), 1L)))
        }
        val impressionsValue: OuroborosValue = RawImpressions(
          impressions = 1L,
          approxUniqueUrls = urlHllOption,
          urlCounts = Some(embedCountSmMonoid.create((referUrl, 1L))),
          urlDates = Some(embedDateSmMonoid.create((referUrl, timestamp))),
          frameUrls = widgetFrameUrlsOption
        ).as[OuroborosValue]
        Seq(
          (OuroborosKey.ImpressionsKey(ImpressionsKey(tweetId.longValue, timeBucketOption)), impressionsValue),
          (OuroborosKey.TopTweetsKey(TopTweetsKey(timeBucketOption)), topTweetsValue)
        )
      }
    }.sumByKey(store)
      .set(MonoidIsCommutative(true))
}
```

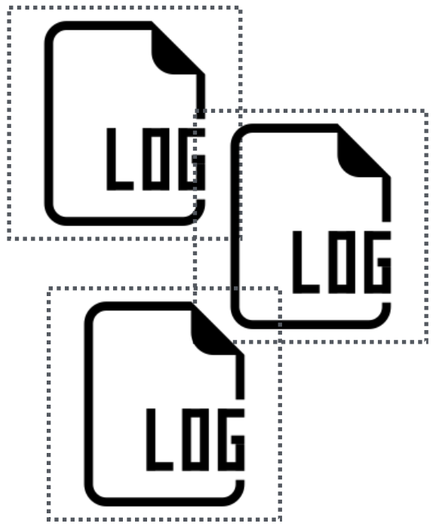
Filter Events

Generate KV Pairs

Sum into Store

- It's about the Monoid (algebraic aggregation)
- Still (too) complicated and hard for non-data infras engineers and non-engineers
- No backend storage support
- No data exploration plan

REAL-TIME PROCESSING + BATCH VALIDATION + DATA EXPLORATION



Aggregation
...
Storage

Common Tasks

Public Safety

Data Analyst
ML/DL Model

Applications

Data Exploration
...
Auditing
...
Validation

Common Tasks

Those are missing!

REAL-TIME PROCESSING + BATCH VALIDATION + DATA EXPLORATION

Batch

- Scalding
- Spark
- GCP Dataflow
- …

Real-time

- Heron
- Eventbus
- Kafka Streams
- Beam
- …

Persistent Storages

- Manhattan
- RDBMS
- Vertica
- HDFS

Query Service

- Similar among apps

#CHALLENGES

For other Engineering/Non-Engineering Teams:

- Research of the optimal solution for their tasks – batch job runners, streaming techniques, backend storages, data exploration tools, etc.
- Stressful maintenance at Twitter's traffic level
- Auditing/Validation/Backfill of the results

For data infrastructure engineering team (us):

- We can't support all the teams for their different needs but with much in common

We'd like to reduce the pain on both sides!

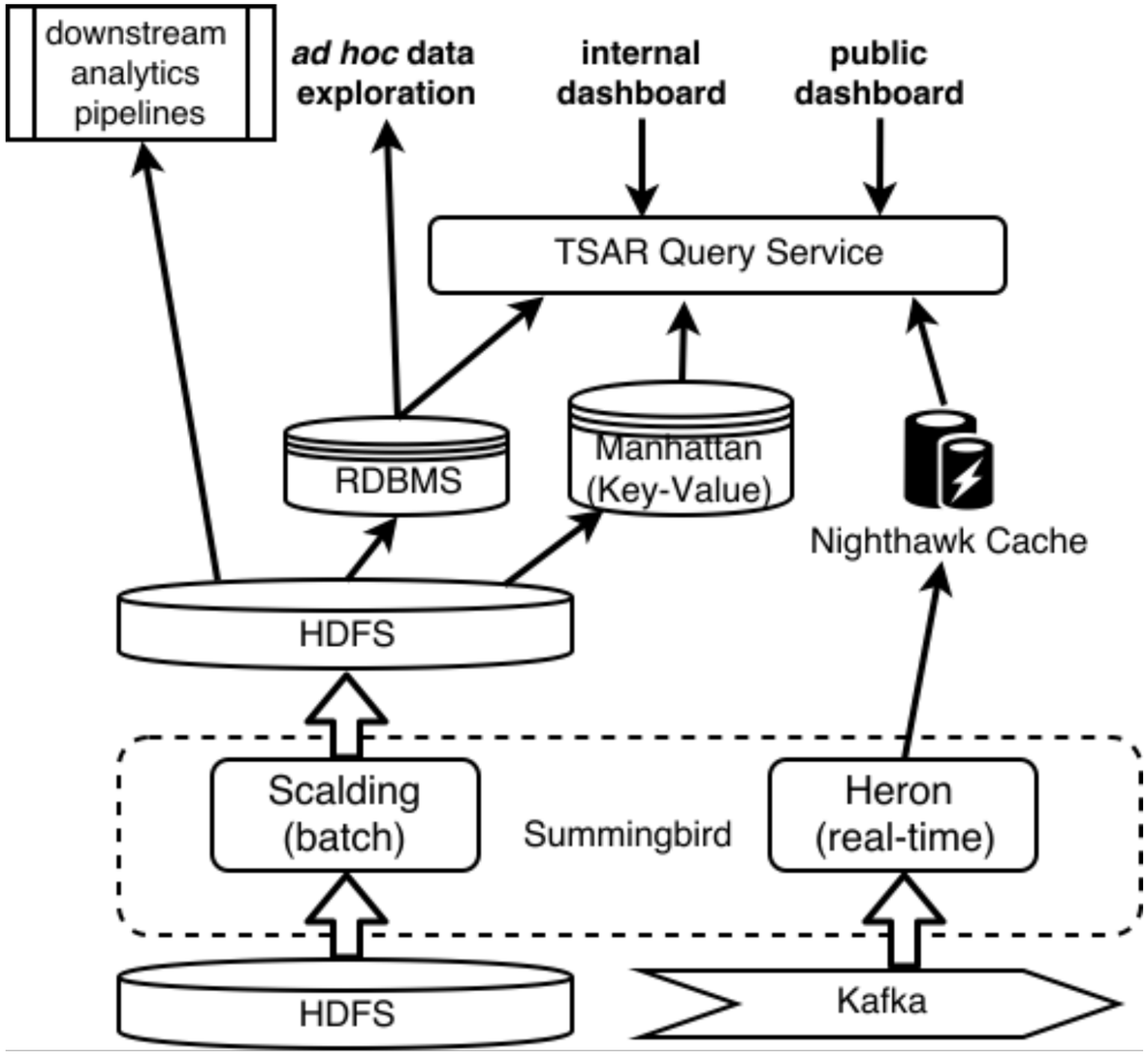
How do we tackle the challenges?

#TSAR

*T*ime*S*eries*A*ggregato*R*

- is Domain Specific Language (DSL)
- builds on top of SummingBird
- incorporates Backend storage options (more complete end-to-end solution)
- comes with Tooling - http/thrift query service, deployment script, easy backfill
- is Easy enough for (almost) everyone at Twitter

#ARCHITECTURE



Case Study: Tweets Engagement

A MINIMAL TSAR PROJECT

#EXAMPLE – TWEETS ENGAGEMENTS


```
struct EngagementAttributes {
    1: optional i64 client_application_id,
    2: optional EngagementType engagement_type,
    3: optional i64 user_id
}
```

#EXAMPLE - TWEETS ENGAGEMENTS

```
aggregate {

    onKeys(
        (clientApplicationId, engagementType)
    )


    produce(Count, Unique(userId))


    sinkTo(Manhattan, NightHawk)


    fromProducer(
        Source.map {
            (
                e.timestamp,
                EngagementAttributes(
                    Some(clientApplicationId),
                    Some(engagementType),
                    Some(userId)
                )
            )
        }
    )
}
```

Dimensions you aggregate on

Metrics

Sinks

Convert events to your schema

Scala Tsar job

#EXAMPLE - TWEETS ENGAGEMENTS

```
aggregate {

    onKeys(
        (clientApplicationId, engagementType),
        (clientApplicationId)
    )

    produce(Count, Unique(userId), Sum)


    sinkTo(Manhattan, NightHawk, Vertica)
    fromProducer(
        Source.map {
            (
                e.timestamp,
                EngagementAttributes(
                    Some(clientApplicationId),
                    Some(engagementType),
                    Some(userId)
                )
            )
        }
    )
}
```

Scala Tsar job

Painless Expansion

#EXAMPLE – TWEETS ENGAGEMENTS

```
Config(
  base = Base(
    namespace             = 'tsar-example',
    name                  = ‘tweets-interaction-counter’,
    user                  = 'tsar-shared',
    thriftAttributesName  = 'TweetAttributes',
    origin                = ‘2018-05-15 00:00:00 UTC',

    jobclass              = 'com.twitter.examples.InteractionCounterJob',

    outputs               =  [
      Output(sink = Sink.IntermediateThrift, width = 1 * Day),
      Output(sink = Sink.Manhattan, width = 1 * Day)
      Output(sink = Sink.Vertica, width = 1 * Day)
    ],
    ...
```

Configuration File

Output datastores && Time granularities for aggregation

AFTER DEPLOYMENT…

- Generate deploy meta-data packaged with your job and logged to Zookeeper
- Compile and bundle your job using pants
- Upload the code to packer
- Auto-generate aurora configuration files
- Deploy a batch job
- Deploy a realtime job
- Deploy a combined http/thrift query service
- Create or update DB tables and views
- Create alerts and viz charts
- Set up anomaly detection

WHAT DO USERS *NOT* SPECIFY?

1) How to represent the schema in RDBMS / Manhattan
2) How to represent the aggregated data
3) How to perform the aggregation
4) How to locate and connect to underlying services (Hadoop, Heron, Manhattan, ...)

LAMBDA OR KAPPA?

A combined solution:

- *Lambda*
  - It has both batch and realtime components
- *Kappa*
  - The users (other developers at Twitter) write one set of code

What's behind the scenes?

DESIGN CONSIDERATIONS

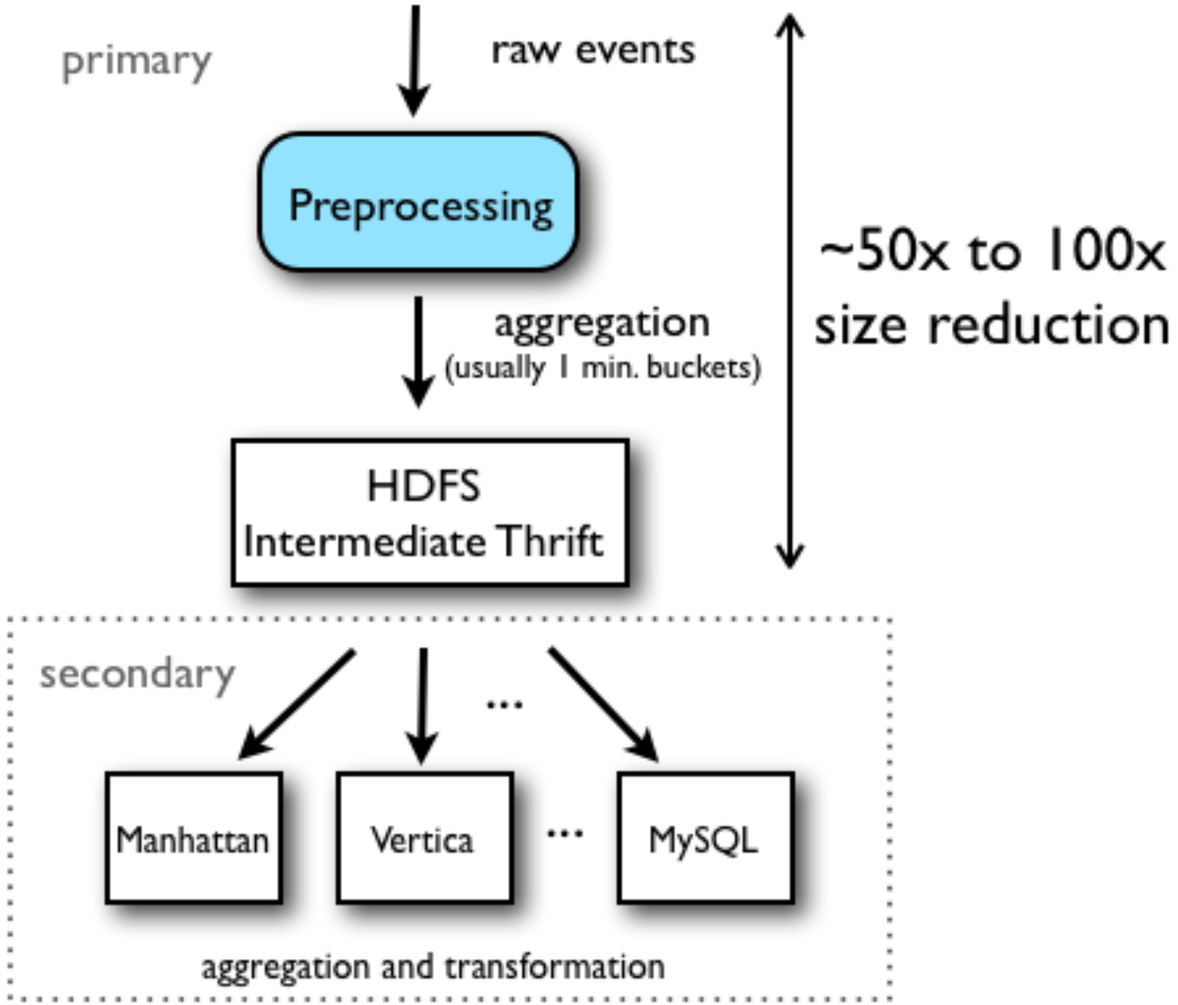
Answers:

- How do we coordinate schemas to keep all physical representations consistent?
  - ***Unified schema architecture generated from thrift schema***
- How do we provide support for flexible schema evolution?
  - ***Separation of event production from event aggregation***
- How do customers easily consume the data?
  - ***Automatically generated http/thrift query service***

PRIMITIVE AND DERIVED METRICS

- Primitive Metrics: Metrics that can be added directly together
  - e.g. Count, Sum
- Derived Metrics: The opposite
  - e.g. Unique, Percentile
- Derived Metrics are computed from Primitive Metrics:
  - e.g. Average = Sum / Count
  - Users don’t need to specify metrics as primitive or derived

PRIMARY AND SECONDARY BATCH JOBS



REAL-TIME WRITE CONSISTENCY AND HOTKEYS

- Sometimes Counter only support Long type
- What about other monoid types? e.g. Double, List
- Tsar solves this by assigning every aggregation key *K* (at compile time) to a unique node in the corresponding Heron topology. That node then has mastership over *K*, and it is guaranteed that no other nodes in the topology will update the value of *K*. (276/280)
- What about Hotkeys then?
- Pre-Aggregation with events/time intervals

#Takeaways

“PLUMBING” WORKS MAKE OTHERS’ LIVES EASIER

#ThankYou