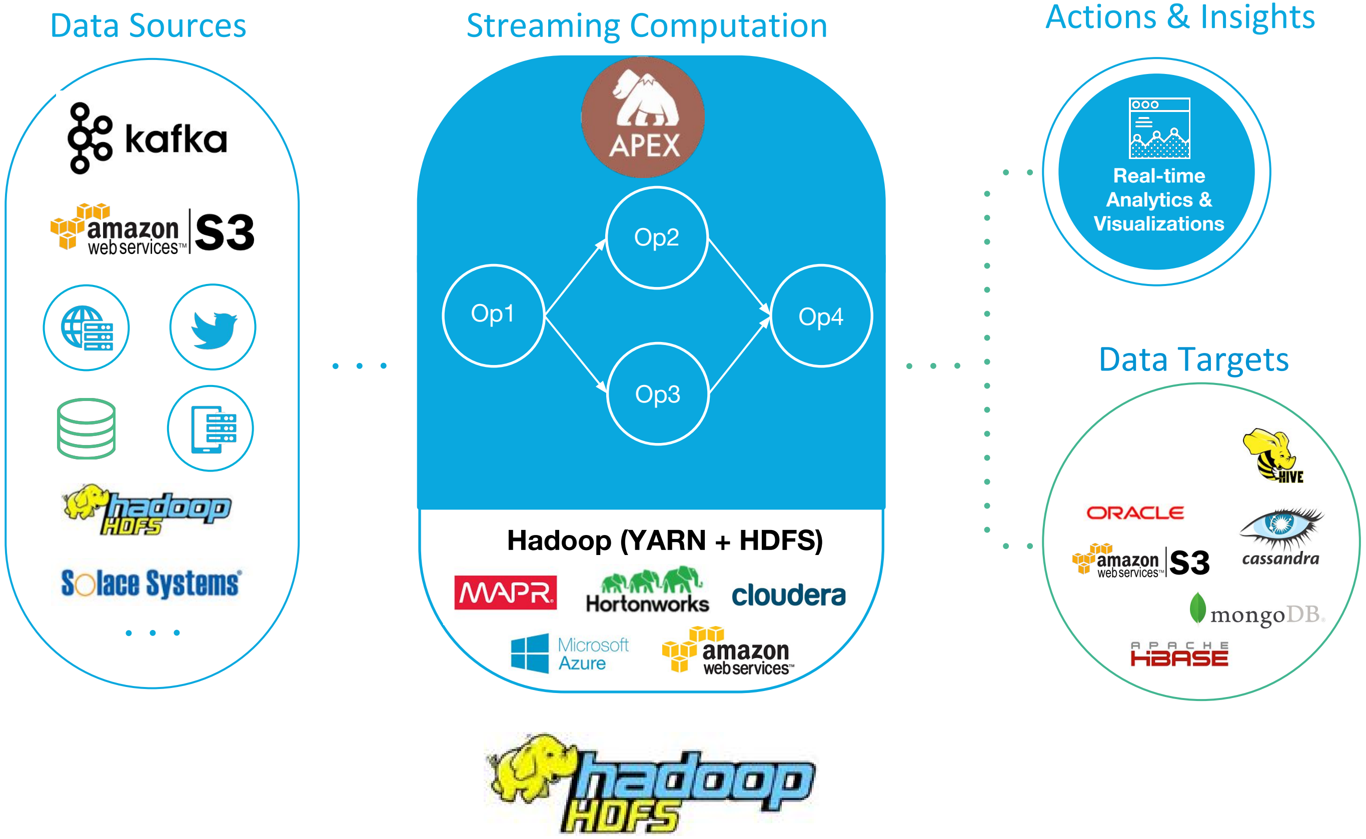
Dimensions Computation With Apache Apex

Devendra Tagare <devtagare@gmail.com>
Data Engineer @ DataTorrent Inc
Committer @ Apache Software Foundation for Apex
@devtagare
ApacheCon North America, 2017

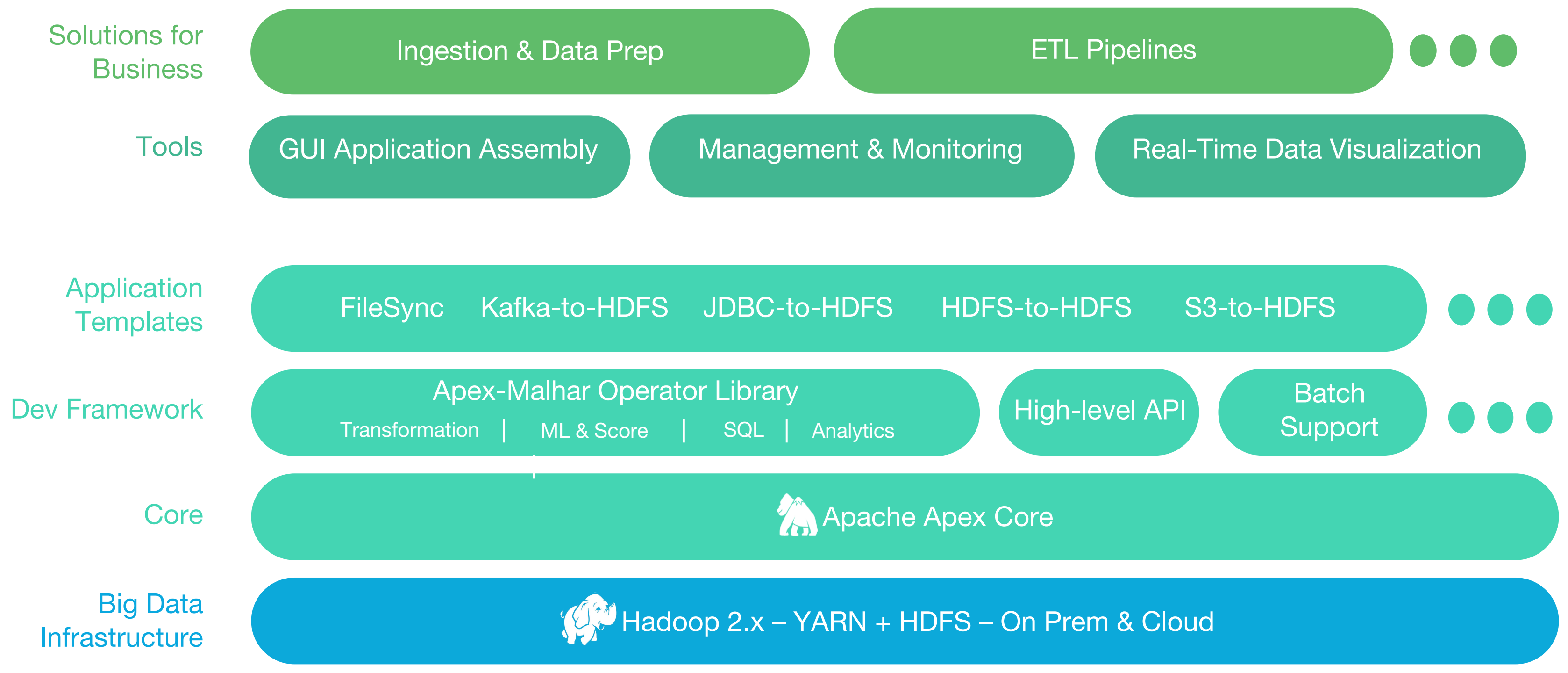
What is Apex ?

- ✓ **Platform and Runtime Engine** - enables development of scalable and fault-tolerant distributed applications for processing streaming and batch data
- ✓ **Highly Scalable** - Scales linearly to billions of events per second with statically defined or dynamic partitioning, advanced locality & affinity
- ✓ **Highly Performant** - In memory computations.Can reach single digit millisecond end-to-end latency
- ✓ **Fault Tolerant** - Automatically recovers from failures - without manual intervention
- ✓ **Stateful** - Guarantees that no state will be lost
- ✓ **YARN Native** - Uses Hadoop YARN framework for resource negotiation
- ✓ **Developer Friendly** - Exposes an easy API for developing *Operators*, which can include any custom business logic written in Java, and provides a Malhar library of many popular operators and application examples.High level API for data scientists/ analysts.

Apex In the Wild

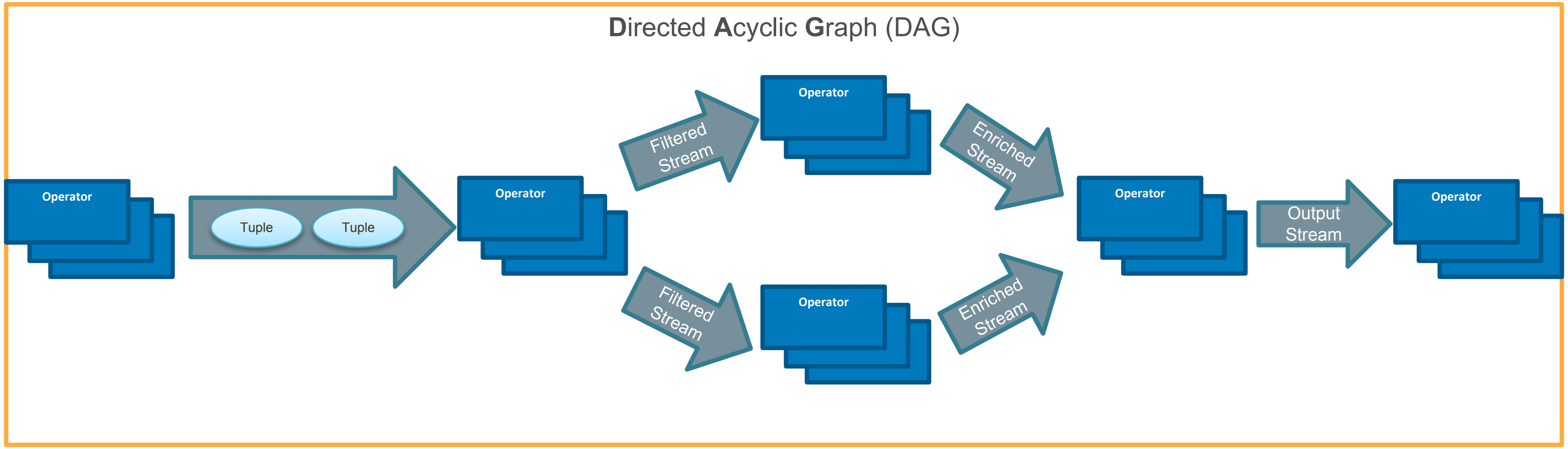


The Apex Ecosystem



4

Application Development Model



- A **Stream** is a sequence of data tuples
- A typical **Operator** takes one or more input streams, performs computations & emits one or more output streams
  - Each Operator is YOUR custom business logic in java, or built-in operator from our open source library
  - Operator has many instances that run in parallel and each instance is single-threaded
- **Directed Acyclic Graph (DAG)** is made up of operators and streams

Stream Locality

- By default operators are deployed in containers (processes) randomly on different nodes across the Hadoop cluster
- Custom locality for streams

Rack local: Data does not traverse network switches

Node local: Data is passed via loopback interface and frees up network bandwidth

Container local: Messages are passed via in memory queues between operators and does not require serialization

Thread local: Messages are passed between operators in a same thread equivalent to calling a subsequent function on the message

Fault Tolerance

- Operator state is check-pointed to a persistent store
  - Automatically performed by engine, no additional work needed by operator
  - In case of failure operators are restarted from checkpoint state
  - Frequency configurable per operator
  - Asynchronous and distributed by default
  - Default store is HDFS
- Automatic detection and recovery of failed operators
  - Heartbeat mechanism
- Buffering mechanism to ensure replay of data from recovered point so that there is no loss of data
- Application master state check-pointed

Processing Guarantees

At-least once

- On recovery data will be replayed from a previous checkpoint
  - Messages will not be lost
  - Default mechanism and is suitable for most applications
- Can be used in conjunction with following mechanisms to achieve exactly-once behavior in fault recovery scenarios
  - Transactions with meta information, Rewinding output, Feedback from external entity, Idempotent operations

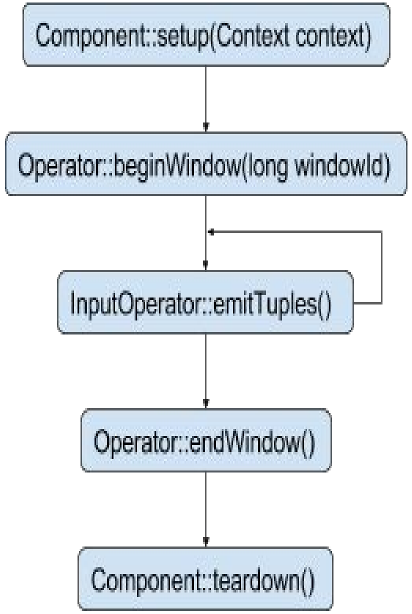
At-most once

- On recovery the latest data is made available to operator
  - Useful in use cases where some data loss is acceptable and latest data is sufficient

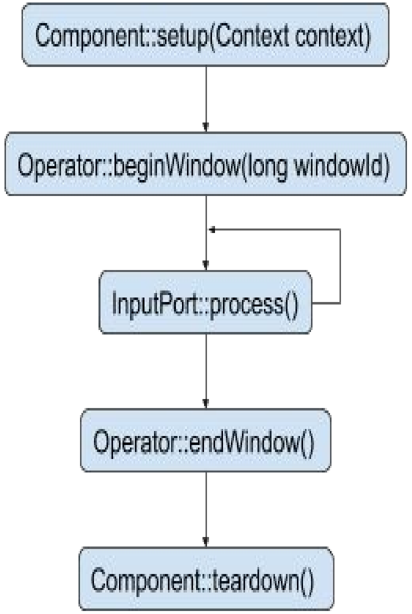
Exactly once

- At least once + state recovery + operator logic to achieve end-to-end exactly once

Apex Operator API


Component::setup(Context context)
Operator::beginWindow(long windowId)
InputOperator::emitTuples()
Operator::endWindow()
Component::teardown()

Flow for Input Adapters


Component::setup(Context context)
Operator::beginWindow(long windowId)
InputPort::process()
Operator::endWindow()
Component::teardown()

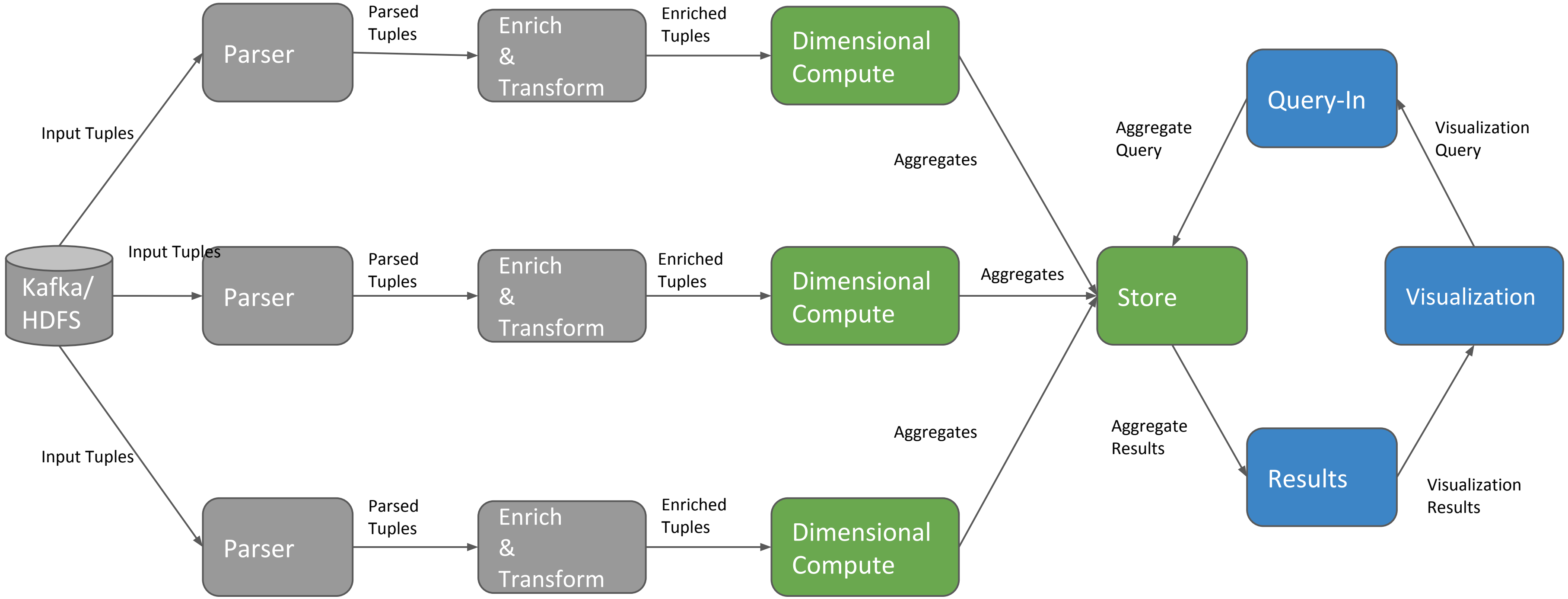
Flow for Generic Operators and Output Adapters

Input Adapters - read from external systems & emit tuples to downstream operators, no input port

Generic Operators - process incoming data received from input adapters or other generic operators.Have both input & output ports

Output Adapters - write to external systems, no output ports

Dimensions Compute Reference Architecture



Dimensional Model - Key Concepts

Metrics : pieces of information we want to collect statistics about.

Dimensions : variables which can impact our measures.

Combinations : set of dimensions for which one or metric would be aggregated.They are sub-sets of dimensions.

Aggregations : the aggregate function eg.. SUM, TOPN, Standard deviation.

Time Buckets : Time buckets are windows of time. Aggregations for a time bucket are comprised only of events with a time stamp that falls into that time bucket.

With the managed state and High level api - Windowed operations also supported for fix window, sliding window, session window for event time, system time, ingestion time.

Example : Ad-Tech : aggregate over key dimensions for revenue metrics

Dimensions - campaignId, advertiserId, time

Metrics - Cost, revenue, clicks, impressions

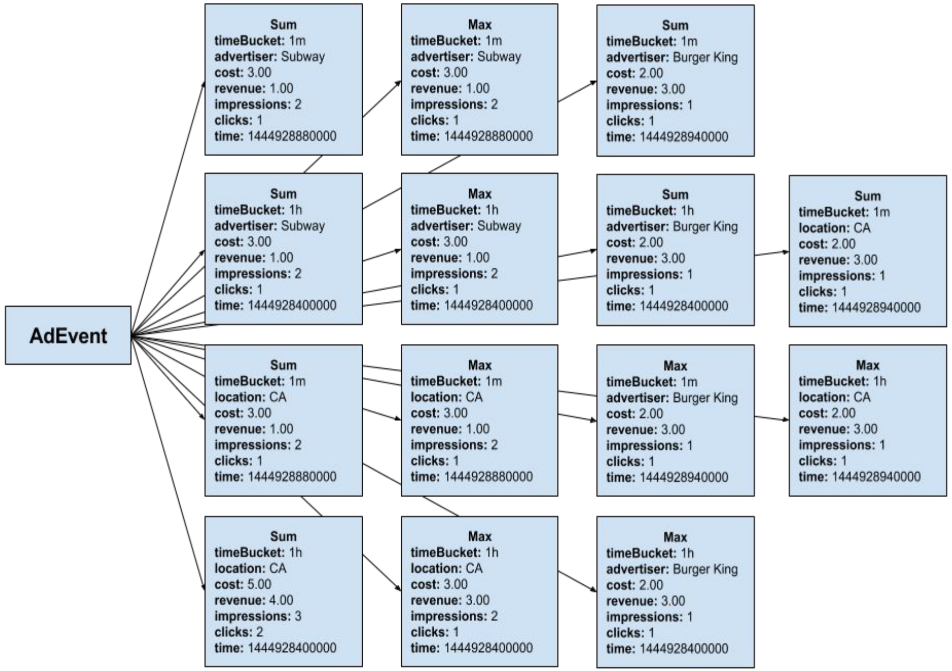
Aggregate functions -SUM,AM etc..

Combinations :

1. campaignId x time - cost,revenue
2. advertiser - revenue, impressions
3. campaignId x advertiser x time - revenue, clicks, impressions

Phases of Dimensional Compute

Aggregations in reality…..


AdEvent
Sum timeBucket: 1m advertiser: Subway cost: 3.00 revenue: 1.00 impressions: 2 clicks: 1 time: 1444928880000
Max timeBucket: 1m advertiser: Subway cost: 3.00 revenue: 1.00 impressions: 2 clicks: 1 time: 1444928880000
Sum timeBucket: 1m advertiser: Burger King cost: 2.00 revenue: 3.00 impressions: 1 clicks: 1 time: 1444928940000
Sum timeBucket: 1h advertiser: Subway cost: 3.00 revenue: 1.00 impressions: 2 clicks: 1 time: 1444928400000
Max timeBucket: 1h advertiser: Subway cost: 3.00 revenue: 1.00 impressions: 2 clicks: 1 time: 1444928400000
Sum timeBucket: 1h advertiser: Burger King cost: 2.00 revenue: 3.00 impressions: 1 clicks: 1 time: 1444928400000
Sum timeBucket: 1m location: CA cost: 2.00 revenue: 3.00 impressions: 1 clicks: 1 time: 1444928940000
Sum timeBucket: 1m location: CA cost: 3.00 revenue: 1.00 impressions: 2 clicks: 1 time: 1444928880000
Max timeBucket: 1m location: CA cost: 3.00 revenue: 1.00 impressions: 2 clicks: 1 time: 1444928880000
Max timeBucket: 1m advertiser: Burger King cost: 2.00 revenue: 3.00 impressions: 1 clicks: 1 time: 1444928940000
Max timeBucket: 1h location: CA cost: 2.00 revenue: 3.00 impressions: 1 clicks: 1 time: 1444928940000
Sum timeBucket: 1h location: CA cost: 5.00 revenue: 4.00 impressions: 3 clicks: 2 time: 1444928400000
Max timeBucket: 1h location: CA cost: 3.00 revenue: 3.00 impressions: 2 clicks: 1 time: 1444928400000
Max timeBucket: 1h advertiser: Burger King cost: 2.00 revenue: 3.00 impressions: 1 clicks: 1 time: 1444928400000

Why break dimensional compute into stages ?

Aggregate footprint in memory generally rises exponentially over time

Scalable implementations of dimensions compute need to handle 100K+ event/sec.

Phases of dimensions compute

The pre-aggregation phase

The unification phase

The aggregation storage phase

The Pre-aggregation phase

Unique Aggregates : Dimensions Computation to scale by reducing the number of events entering the system
Example : 'n' events flowing through the system actually translate to a lower # unique aggregates
eg 500,000 adEvents flowing through the system actually translate to around 10,000 aggregates due to repeating keys.

Partitioning : use partitioning to scale up the dimensional compute.
Example : If a partition can handle 500,000 events/second, then 8 partitions would be able to handle 4,000,000 events/second which are effectively combined into 80,00 aggregates/second

Problem of the Incomplete Aggregations ?
Aggregate values from previous batches not factored in - corrected in the Aggregation Storage phase.
Different partitions may share the say key and time buckets - partial aggregates - corrected in Unification phase.

Setting up the Pre-Aggregation phase of Dimensions Computation involves configuring a Dimension Computation operator - DimensionsComputationFlexibleSingleSchemaPOJO

The Dimensional Model

```
{"keys":[{"name":"campaignId","type":"integer"},
       {"name":"adId","type":"integer"},
       {"name":"creativeId","type":"integer"},
       {"name":"publisherId","type":"integer"},
       {"name":"adOrderId","type":"integer"}],
 "timeBuckets":["1h","1d"],
 "values":
[{"name":"impressions","type":"integer","aggregators":["SUM"]}
,
  {"name":"clicks","type":"integer","aggregators":["SUM"]},
  {"name":"revenue","type":"integer"}],
 "dimensions":
  [{"combination":["campaignId","adId"]},
  {"combination":["creativeId","campaignId"]},
  {"combination":["campaignId"]},
  {"combination":["publisherId","adOrderId","campaignId"],
"additionalValues":["revenue:SUM"]}]
}
```

Ad Event

```
public AdEvent(String publisherId,
                           String campaignId
                            String location,
                            double cost,
                            double revenue,
                            long impressions,
                            long clicks,
                            long time….)
      {
        this.publisherId = publisherId;
        this.campaignId = campaignId;
        this.location = location;
        this.cost = cost;
        this.revenue = revenue;
        this.impressions = impressions;
        this.clicks = clicks;
        this.time = time;
        ….
      }
      /* Getters and setters go here */
```

The Unification Phase

Combines outputs - combines the outputs of all the partitions in the Pre-Aggregation phase into a single single stream which can be passed on to the storage phase

Why combine ?

To reduce the number of aggregations even further ~ lower memory footprint, higher throughput

This is because the aggregations produced by different partitions which share the same key and time bucket can be combined to produce a single aggregation ~ completeness for point to point query

Example : if the Unification phase receives 80,000 aggregations/second, you can expect 20,000 aggregations/second after unification.

Implementation : Add a unifier that can be set on your dimensions computation operator,

```
dimensions.setUnifier(new DimensionsComputationUnifierImpl<InputEvent,
Aggregate>());
```

The Aggregation Storage Phase

Aggregation Persistence : Aggregations are persisted to HDFS using HDHT.

Dimensions Store persists aggregates and serves the below functions

Functions as a storage so that aggregations can be retrieved for visualization.

Functions as a storage allowing aggregations to be combined with incomplete aggregates produced by Unification.

Visualization

The Dimensions Store allows you to visualize your aggregations over time. This is done by allowing queries and responses to be received from and sent to the UI via websocket.

Aggregation

The store produces complete aggregations by combining the incomplete aggregations received from the Unification stage with aggregations persisted to HDFS.

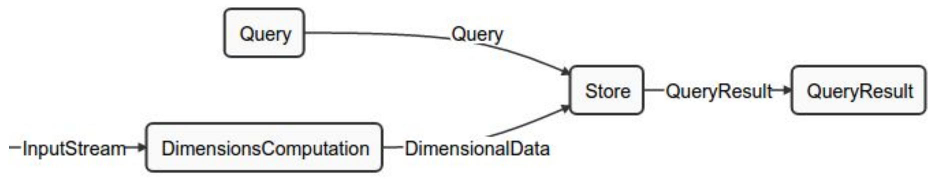
Why have the previous phases ?

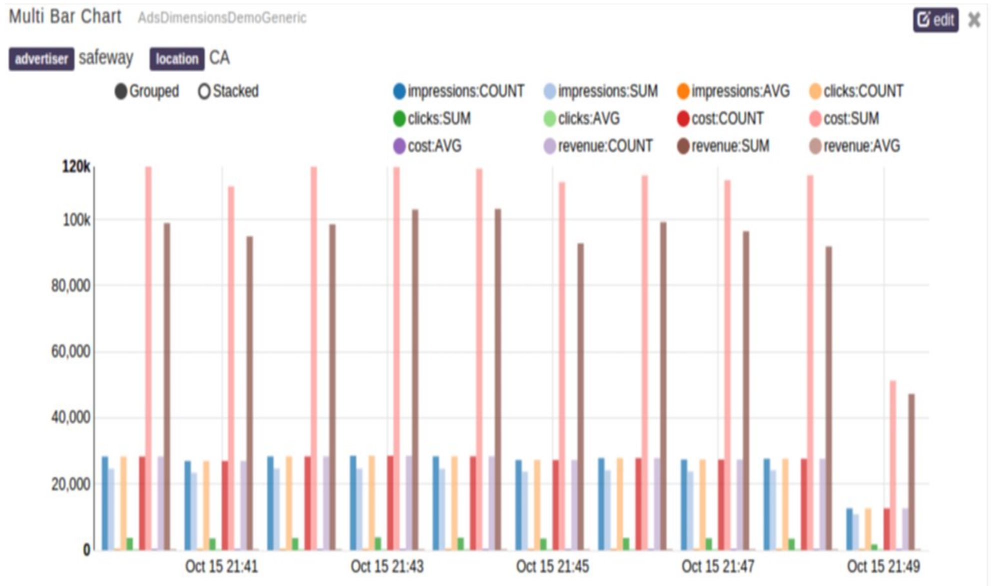
Dimensions Store is I/O intensive, and may cause bottle-necks.

Previous phases reduce the cardinality of events so that the Store will always have lesser # events.

Other variants & the new way : Use managed state instead of HDHT.

Visualization with Apex





AdEvents over time

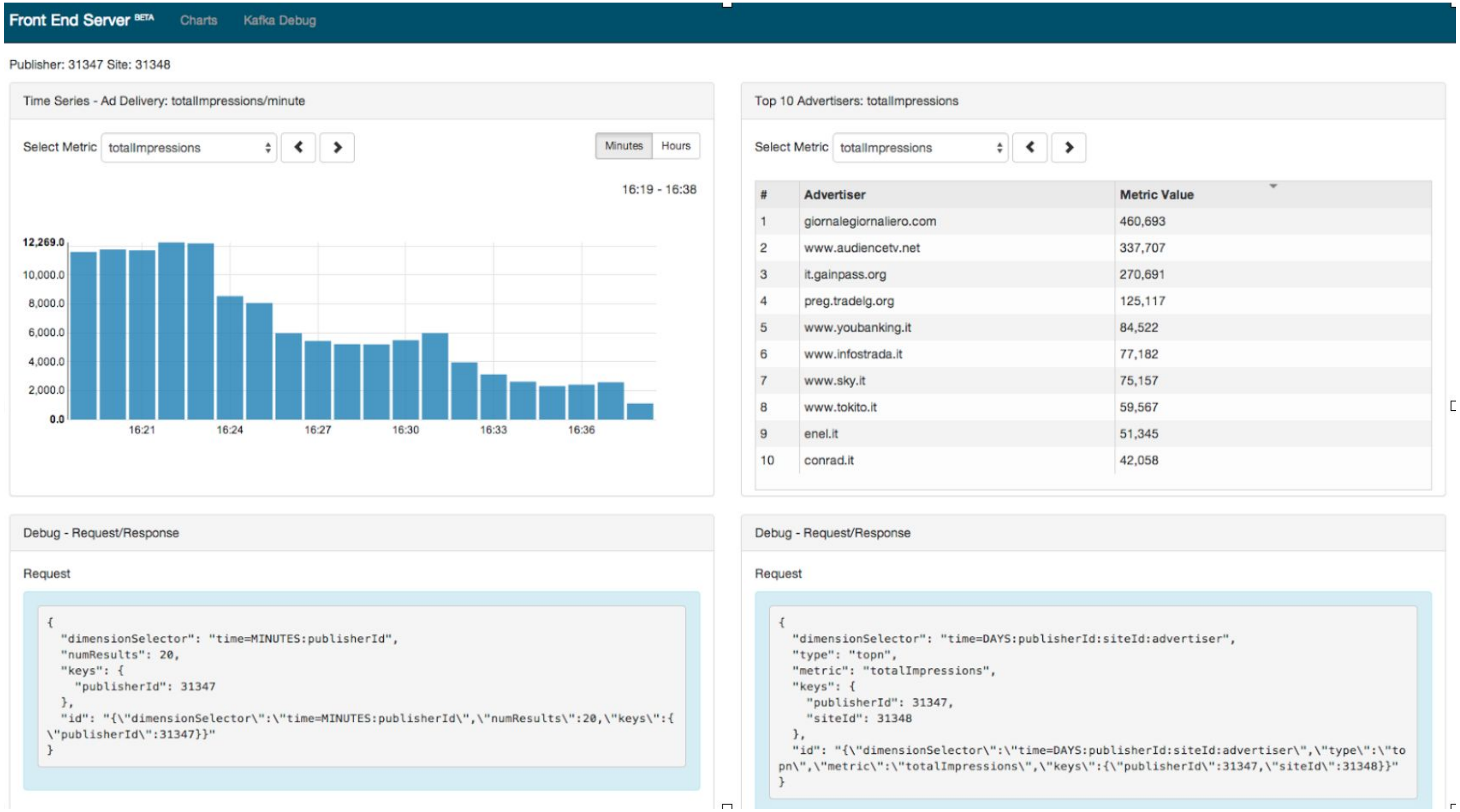
Query

Browser creates a websocket connection with the pubsub server hosted by a webserver.
UI Widgets based on the Malhar angular dashboard can send queries to the pubsub server via this connection to a specific topic.
These queries are parsed by the Query operator and passed onto DimensionsStore to fetch data from HDHT Store.

QueryResult

The QueryResult operator gets the result from the DimensionsStore operator for a given query, formats and renders it to the widget.

Sample Visualization



Publisher: 31347 Site: 31348

Time Series - Ad Delivery: totalImpressions/minute

Select Metric totalImpressions

Top 10 Advertisers: totalImpressions

Select Metric totalImpressions

#	Advertiser	Metric Value
1	giornalegiornaliero.com	460,693
2	www.audiencetv.net	337,707
3	it.gainpass.org	270,691
4	preg.tradelg.org	125,117
5	www.youbanking.it	84,522
6	www.infostrada.it	77,182
7	www.sky.it	75,157
8	www.tokito.it	59,567
9	enel.it	51,345
10	conrad.it	42,058

Debug - Request/Response

Request

```
{
  "dimensionSelector": "time=MINUTES:publisherId",
  "numResults": 20,
  "keys": {
    "publisherId": 31347
  },
  "id": "{\"dimensionSelector\":\"time=MINUTES:publisherId\",\"numResults\":20,\"keys\":{
\"publisherId\":31347}}"
}
```

Debug - Request/Response

Request

```
{
  "dimensionSelector": "time=DAYS:publisherId:siteId:advertiser",
  "type": "topn",
  "metric": "totalImpressions",
  "keys": {
    "publisherId": 31347,
    "siteId": 31348
  },
  "id": "{\"dimensionSelector\":\"time=DAYS:publisherId:siteId:advertiser\",\"type\":\"to
pn\",\"metric\":\"totalImpressions\",\"keys\":{\"publisherId\":31347,\"siteId\":31348}}"
}
```

Q & A

Thank You !!!

Resources

Apache Apex - http://apex.apache.org/

References : http://docs.datatorrent.com/

Subscribe to forums : Apex - http://apex.apache.org/community.html

Download - http://apex.apache.org/downloads

Twitter : @ApacheApex; Follow - https://twitter.com/apacheapex

Meetups - http://meetup.com/topics/apache-apex