Druid pour l'analyse de données en temps réel

Yann Esposito

7 Avril 2016

Abstract

Druid expliqué rapidement, pourquoi, comment.

Contents

1 Intro

1.1 Plan

- Introduction; why?
- How?

1.2 Experience

- Real Time Social Media Analytics

1.3 Real Time?

- Ingestion Latency: seconds
- Query Latency: seconds

1.4 Demande

- Twitter: `20k msg/s`, `1msg = 10ko` pendant 24h
- Facebook public: 1000 à 2000 msg/s en continu

1.5 En pratique

- Twitter: 400 msg/s en continu, pics à 1500

1.6 Origine (PHP)

1.7 Introduction

- Traitement de donnée gros volume + faible latence
- Typiquement `pulse`

DEMO

1.8 Pre Considerations

Discovered vs Invented

1.9 Try to conceptualize (events)

Scalable + Real Time + Fail safe

- timeseries
- alerting system
- top N
- etc…

1.10 In the End

Druid concepts are always emerging naturally

2 Druid

2.1 Who

Metamarkets

2.2 Goal

> Druid is an open source store designed for real-time exploratory analytics on large data sets.
>
> hosted dashboard that would allow users to arbitrarily explore and visualize event streams.

2.3 Concepts

- Column-oriented storage layout
- distributed, shared-nothing architecture
- advanced indexing structure

2.4 Features

- fast aggregations
- flexible filters
- low latency data ingestion

arbitrary exploration of billion-row tables tables with sub-second latencies

2.5 Storage

- Columnar
- Inverted Index
- Immutable Segments

2.6 Columnar Storage

2.7 Index

- Values are dictionary encoded

```
{"USA" 1, "Canada" 2, "Mexico" 3, ...}
```

- Bitmap for every dimension value (used by filters)

```
"USA" -> [0 1 0 0 1 1 0 0 0]
```

- Column values (used by aggergation queries)

```
[2,1,3,15,1,1,2,8,7]
```

2.8 Data Segments

- Per time interval
- skip segments when querying
- Immutable
- Cache friendly
- No locking
- Versioned
- No locking
- Read-write concurrency

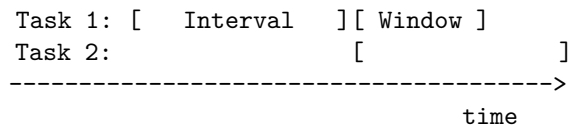
2.9 Real-time ingestion

- Via Real-Time Node and Firehose
- No redundancy or HA, thus not recommended
- Via Indexing Service and Tranquility API
- Core API
- Integration with Streaming Frameworks
- HTTP Server
- **Kafka Consumer**

2.10 Batch Ingestion

- File based (HDFS, S3, …)

2.11 Real-time Ingestion



Minimum indexing slots =
Data Sources × Partitions × Replicas × 2

3 Querying

3.1 Query types

- Group by: group by multiple dimensions
- Top N: like grouping by a single dimension
- Timeseries: without grouping over dimensions
- Search: Dimensions lookup
- Time Boundary: Find available data timeframe
- Metadata queries

3.2 Tip

- Prefer `topN` over `groupBy`
- Prefer `timeseries` over `topN`
- Use limits (and priorities)

3.3 Query Spec

- Data source
- Dimensions
- Interval
- Filters
- Aggergations
- Post Aggregations
- Granularity
- Context (query configuration)
- Limit

3.4 Example(s)

TODO

3.5 Caching

- Historical node level
- By segment
- Broker Level
- By segment and query
- `groupBy` is disabled on purpose!
- By default - local caching

3.6 Load Rules

- Can be defined
- What can be set

4 Components

4.1 Druid Components

- Real-time Nodes
- Historical Nodes
- Broker Nodes
- Coordinator
- For indexing:
- Overlord
- Middle Manager
- Deep Storage
- Metadata Storage
- Load Balancer
- Cache

4.2 Coordinator

Manage Segments

4.3 Real-time Nodes

- Pulling data in real-time
- Indexing it

4.4 Historical Nodes

- Keep historical segments

4.5 Overlord

- Accepts tasks and distributes them to middle manager

4.6 Middle Manager

- Execute submitted tasks via Peons

4.7 Broker Nodes

- Route query to Real-time and Historical nodes
- Merge results

4.8 Deep Storage

- Segments backup (HDFS, S3, …)

5 Considerations & Tools

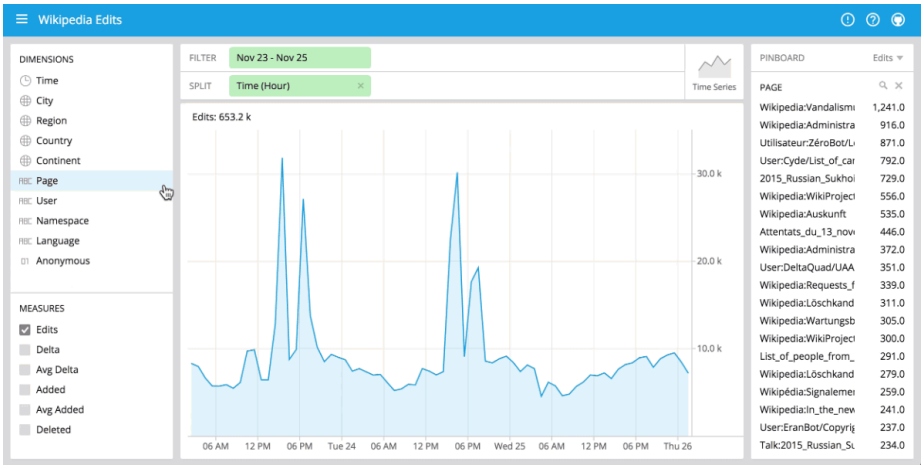
5.1 When *not* to choose Druid

- Data is not time-series
- Cardinality is *very* high
- Number of dimensions is high
- Setup cost must be avoided

5.2 Graphite (metrics)

Graphite

5.3 Pivot (exploring data)



Pivot

5.4 Caravel (exploring data)

Caravel