Druid for real-time analysis

Yann Esposito

7 Avril 2016

Abstract

Druid explained with high altitude point of view

Contents

1 Druid the Sales Pitch **4**

2 Intro **4**

2.1 Experience 4

2.2 Real Time? 4

2.3 Demand 4

2.4 Reality 5

3 Origin (PHP) **5**

4 1st Refactoring (Node.js) **5**

5 Return of Experience **6**

6 Return of Experience **6**

7 2nd Refactoring **6**

8 2nd Refactoring (FTW!) **7**

9 2nd Refactoring return of experience **8**

10 Demo **8**

11 Pre Considerations **8**

11.1 Discovered vs Invented 8

11.2 In the End 8

12 Druid **8**

12.1 Who? 8

12.2 Goal 9

12.3 Concepts 9

12.4 Key Features 9

12.5 Right for me? 9

13 High Level Architecture **9**

13.1 Inspiration 9

13.2 Index / Immutability 9

13.3 Storage 10

13.4 Specialized Nodes 10

14 Druid vs X **10**

14.1 Elasticsearch 10

14.2 Key/Value Stores (HBase/Cassandra/OpenTSDB) 10

14.3 Spark 10

14.4 SQL-on-Hadoop (Impala/Drill/Spark SQL/Presto) 10

15 Data **11**

15.1 Concepts 11

16 Roll-up **11**

16.1 Example 11

16.2 as SQL 11

17 Sharding **11**

17.1 Segments 11

17.2 Core Data Structure 12

17.3 Dictionary 12

17.4 Columnn Data 12

17.5 Bitmaps 12

18 Data **13**

18.1 Indexing Data 13

18.2 Loading data 13

18.3 Querying the data 13

18.4 Columnar Storage 13

18.5 Index 13

18.6 Data Segments 13

18.7 Real-time ingestion 14

18.8 Batch Ingestion 14

18.9 Real-time Ingestion 14

19 Querying **14**

19.1 Query types 14

19.2 Tip 15

19.3 Query Spec 15

19.4 Example(s) 15

19.5 Caching 15

19.6 Load Rules 15

20 Components **16**

20.1 Druid Components 16

20.2 Coordinator 16

20.3 Real-time Nodes 16

20.4 Historical Nodes 16

20.5 Overlord 16

20.6 Middle Manager 16
20.7 Broker Nodes 17
20.8 Deep Storage 17

21 Considerations & Tools **17**

21.1 When *not* to choose Druid 17
21.2 Graphite (metrics) 17
21.3 Pivot (exploring data) 18
21.4 Caravel (exploring data) 18

1 Druid the Sales Pitch

- Sub-Second Queries
- Real-time Streams
- Scalable to Petabytes
- Deploy Anywhere
- Vibrant Community (Open Source)

- Ideal for powering user-facing analytic applications
- Deploy anywhere: cloud, on-premise, integrate with Haddop, Spark, Kafka, Storm, Samza

2 Intro

2.1 Experience

- Real Time Social Media Analytics

2.2 Real Time?

- Ingestion Latency: seconds
- Query Latency: seconds

2.3 Demand

- Twitter: `20k msg/s`, `1msg = 10ko` during 24h
- Facebook public: 1000 to 2000 msg/s continuously
- Low Latency

2.4 Reality

- Twitter: 400 msg/s continuously, burst to 1500
- Facebook: 1000 to 2000 msg/s

3 Origin (PHP)

4 1st Refactoring (Node.js)

- Ingestion still in PHP
- Node.js, Perl, Java & R for sentiment analysis
- MongoDB
- Manually made time series (Incremental Map/Reduce)
- Manually coded HyperLogLog in js

5 Return of Experience

6 Return of Experience

- Ingestion still in PHP (600 msg/s max)
- Node.js, Perl, Java (10 msg/s max)

7 2nd Refactoring

- Haskell
- Clojure / Clojurescript
- Kafka / Zookeeper
- Mesos / Marathon
- Elasticsearch
- **Druid**

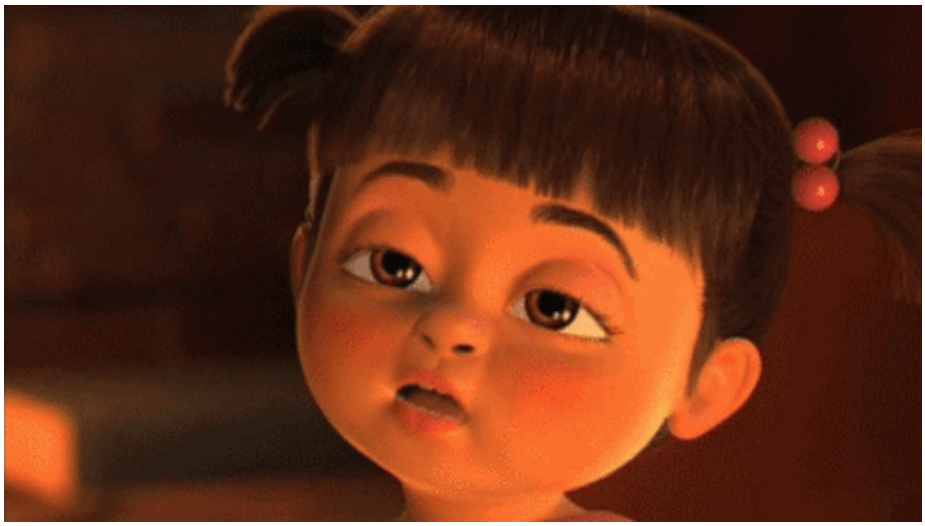


Figure 1: Too Slow, Bored

8 2nd Refactoring (FTW!)

9 2nd Refactoring return of experience

- No limit, everything is scalable
- High availability
- Low latency: Ingestion & User faced querying
- Cheap if done correctly

Thanks Druid!

10 Demo

- Low Latency High Volume of Data Analysis
- Typically `pulse`

DEMO Time

11 Pre Considerations

11.1 Discovered vs Invented

Try to conceptualize a s.t.

- Ingest Events
- Real-Time Queries
- Scalable
- Highly Available

Analytics: timeseries, alerting system, top N, etc…

11.2 In the End

Druid concepts are always emerging naturally

12 Druid

12.1 Who?

Metamarkets

Powered by Druid

- Alibaba, Cisco, Criteo, eBay, Hulu, Netflix, Paypal…

12.2 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.

12.3 Concepts

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

12.4 Key Features

- Sub-second OLAP Queries
- Real-time Streaming Ingestion
- Power Analytic Applications
- Cost Effective
- High Available
- Scalable

12.5 Right for me?

- require fast aggregations
- exploratory analytics
- analysis in real-time
- lots of data (trillions of events, petabytes of data)
- no single point of failure

13 High Level Architecture

13.1 Inspiration

- Google's BigQuery/Dremel
- Google's PowerDrill

13.2 Index / Immutability

Druid indexes data to create mostly immutable views.

13.3 Storage

Store data in custom column format highly optimized for aggregation & filter.

13.4 Specialized Nodes

- A Druid cluster is composed of various type of nodes
- Each designed to do a small set of things very well
- Nodes don't need to be deployed on individual hardware
- Many node types can be colocated in production

14 Druid vs X

14.1 Elasticsearch

- resource requirement much higher for ingestion & aggregation
- No data summarization (100x in real world data)

14.2 Key/Value Stores (HBase/Cassandra/OpenTSDB)

- Must Pre-compute Result
  - Exponential storage
  - Hours of pre-processing time
- Use the dimensions as key (like in OpenTSDB)
  - No filter index other than range
  - Hard for complex predicates

14.3 Spark

- Druid can be used to accelerate OLAP queries in Spark
- Druid focuses on the latencies to ingest and serve queries
- Too long for end user to arbitrarily explore data

14.4 SQL-on-Hadoop (Impala/Drill/Spark SQL/Presto)

- Queries: more data transfer between nodes
- Data Ingestion: bottleneck by backing store
- Query Flexibility: more flexible (full joins)

15 Data

15.1 Concepts

- **Timestamp column**: query centered on time axis
- **Dimension columns**: strings (used to filter or to group)
- **Metric columns**: used for aggregations (count, sum, mean, etc…)

16 Roll-up

16.1 Example

```
timestamp             page          ... added  deleted
2011-01-01T00:01:35Z  Justin Bieber     10      65
2011-01-01T00:03:63Z  Justin Bieber     15      62
2011-01-01T01:04:51Z  Justin Bieber     32      45
2011-01-01T01:01:00Z  Ke$ha             17      87
2011-01-01T01:02:00Z  Ke$ha             43      99
2011-01-01T02:03:00Z  Ke$ha             12      53
```

```
timestamp             page          ... nb added deleted
2011-01-01T00:00:00Z  Justin Bieber      2 25    127
2011-01-01T01:00:00Z  Justin Bieber      1 32    45
2011-01-01T01:00:00Z  Ke$ha              2 60    186
2011-01-01T02:00:00Z  Ke$ha              1 12    53
```

16.2 as SQL

```
GROUP BY timestamp, page, nb, added, deleted
  :: nb = COUNT(1)
  ,  added = SUM(added)
  ,  deleted = SUM(deleted)
```

In practice can dramatically reduce the size (up to x100)

17 Sharding

17.1 Segments

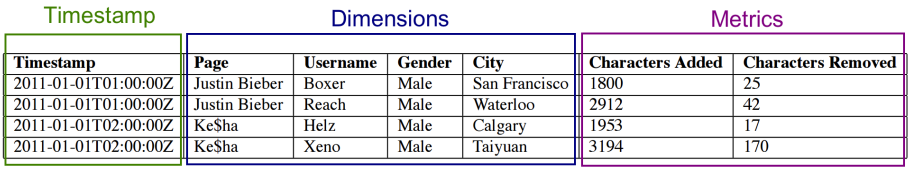
```
sampleData_2011-01-01T01:00:00:00Z_2011-01-01T02:00:00:00Z_v1_0
```

```
2011-01-01T01:00:00Z  Justin Bieber      1 20    45
2011-01-01T01:00:00Z  Ke$ha              1 30    106

sampleData_2011-01-01T01:00:00:00Z_2011-01-01T02:00:00:00Z_v1_0

2011-01-01T01:00:00Z  Justin Bieber      1 12    45
2011-01-01T01:00:00Z  Ke$ha              2 30    80
```

17.2 Core Data Structure



Timestamp	Page	Username	Gender	City	Characters Added	Characters Removed
2011-01-01T01:00:00Z	Justin Bieber	Boxer	Male	San Francisco	1800	25
2011-01-01T01:00:00Z	Justin Bieber	Reach	Male	Waterloo	2912	42
2011-01-01T02:00:00Z	Ke$ha	Helz	Male	Calgary	1953	17
2011-01-01T02:00:00Z	Ke$ha	Xeno	Male	Taiyuan	3194	170

- dictionary
- a bitmap for each value
- a list of the columns values encoded using the dictionary

17.3 Dictionary

```
{ "Justin Bieber": 0
, "Ke$ha": 1
}
```

17.4 Columnn Data

```
[ 0
, 0
, 1
, 1
]
```

17.5 Bitmaps

one for each value of the column

```
value="Justin Bieber": [1,1,0,0]
value="Ke$ha": [0,0,1,1]
```

18 Data

18.1 Indexing Data

- Immutable snapshots of data
- data structure highly optimized for analytic queries
- Each column is stored separately
- Indexes data on a per shard (segment) level

18.2 Loading data

- Real-Time
- Batch

18.3 Querying the data

- JSON over HTTP
- Single Table Operations, no joins.

18.4 Columnar Storage

18.5 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]
```

18.6 Data Segments

- Per time interval
  - skip segments when querying

- Immutable
  - Cache friendly
  - No locking
- Versioned
  - No locking
  - Read-write concurrency

18.7 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**

18.8 Batch Ingestion

- File based (HDFS, S3, …)

18.9 Real-time Ingestion

```
Task 1: [   Interval   ][ Window ]
Task 2:                  [              ]
---------------------------------------->
                                   time
```

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

19 Querying

19.1 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

19.2 Tip

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

19.3 Query Spec

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

19.4 Example(s)

TODO

19.5 Caching

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

19.6 Load Rules

- Can be defined
- What can be set

20 Components

20.1 Druid Components

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

20.2 Coordinator

Manage Segments

20.3 Real-time Nodes

- Pulling data in real-time
- Indexing it

20.4 Historical Nodes

- Keep historical segments

20.5 Overlord

- Accepts tasks and distributes them to middle manager

20.6 Middle Manager

- Execute submitted tasks via Peons

20.7 Broker Nodes

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

20.8 Deep Storage

- Segments backup (HDFS, S3, …)

21 Considerations & Tools

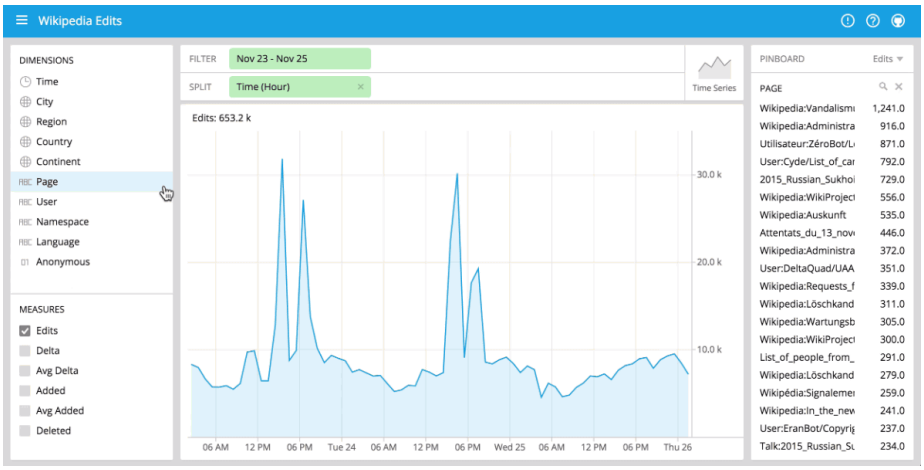
21.1 When *not* to choose Druid

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

21.2 Graphite (metrics)

Graphite

21.3 Pivot (exploring data)



Pivot

21.4 Caravel (exploring data)

Caravel