

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	4
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
15.2 Indexing	11
15.3 Loading	11
15.4 Querying	11
15.5 Segments	11

16 Roll-up	12
16.1 Example	12
16.2 as SQL	12
17 Segments	12
17.1 Sharding	12
17.2 Core Data Structure	13
17.3 Example	13
17.4 Example (multiple matches)	13
17.5 Real-time ingestion	13
17.6 Batch Ingestion	14
17.7 Real-time Ingestion	14
18 Querying	14
18.1 Query types	14
18.2 Example(s)	14
18.3 Result	15
18.4 Caching	15
19 Druid Components	15
19.1 Druid	15
19.2 Also	15
19.3 Coordinator	16
20 When <i>not</i> to choose Druid	16
21 Graphite (metrics)	16
22 Pivot (exploring data)	17
23 Caravel	17
24 Conclusions	18
24.1 Precompute your time series?	18
24.2 Don't reinvent it	18
24.3 Druid way is the right way!	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 Hadoop, 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**

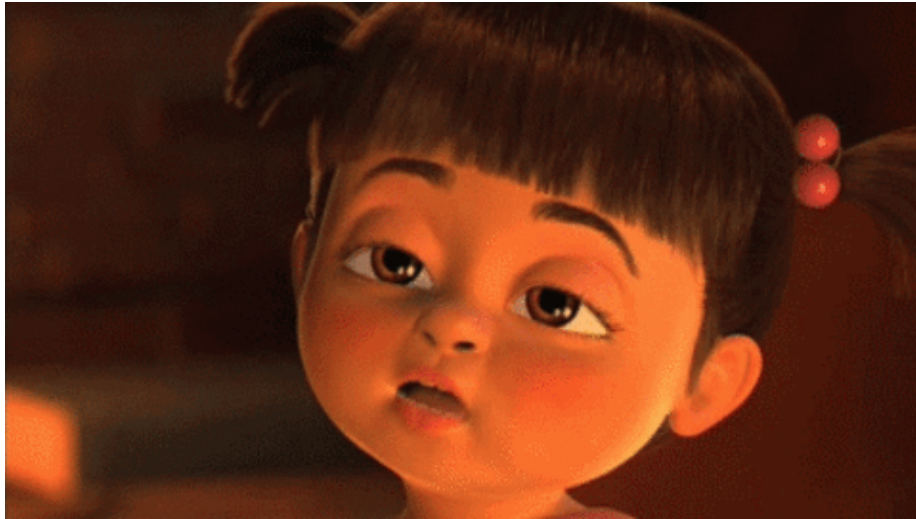


Figure 1: Too Slow, Bored

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...)

15.2 Indexing

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

15.3 Loading

- Real-Time
- Batch

15.4 Querying

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

15.5 Segments

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

16 Roll-up

16.1 Example

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

timestamp	page	...	nb	added	deleted
2011-01-01T00:00:00Z	Cthulhu		2	25	127
2011-01-01T01:00:00Z	Cthulhu		1	32	45
2011-01-01T01:00:00Z	Azatoth		2	60	186
2011-01-01T02:00:00Z	Azatoth		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 Segments

17.1 Sharding

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

timestamp	page	...	nb	added	deleted
2011-01-01T01:00:00Z	Cthulhu		1	20	45
2011-01-01T01:00:00Z	Azatoth		1	30	106

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

timestamp	page	...	nb	added	deleted
2011-01-01T01:00:00Z	Cthulhu		1	12	45
2011-01-01T01:00:00Z	Azatoth		2	30	80

17.2 Core Data Structure

Timestamp	Dimensions				Metrics	
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 Example

```
dictionary: { "Cthulhu": 0
              , "Azatoth": 1 }
```

```
column data: [0, 0, 1, 1]
```

```
bitmaps (one for each value of the column):
value="Cthulhu": [1,1,0,0]
value="Azatoth": [0,0,1,1]
```

17.4 Example (multiple matches)

```
dictionary: { "Cthulhu": 0
              , "Azatoth": 1 }
```

```
column data: [0, [0,1], 1, 1]
```

```
bitmaps (one for each value of the column):
value="Cthulhu": [1,1,0,0]
value="Azatoth": [0,1,1,1]
```

17.5 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**

17.6 Batch Ingestion

- File based (HDFS, S3, ...)

17.7 Real-time Ingestion

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

18 Querying

18.1 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

18.2 Example(s)

```
{ "queryType": "groupBy",
  "dataSource": "druidtest",
  "granularity": "all",
  "dimensions": [],
  "aggregations": [
    { "type": "count", "name": "rows" },
    { "type": "longSum", "name": "imps", "fieldName": "impressions" },
    { "type": "doubleSum", "name": "wp", "fieldName": "wp" }
  ],
  "intervals": ["2010-01-01T00:00/2020-01-01T00"] }
```

18.3 Result

```
[ {
  "version" : "v1",
  "timestamp" : "2010-01-01T00:00:00.000Z",
  "event" : {
    "imps" : 5,
    "wp" : 15000.0,
    "rows" : 5
  }
} ]
```

18.4 Caching

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

19 Druid Components

19.1 Druid

- Real-time Nodes
- Historical Nodes
- Broker Nodes
- Coordinator
- For indexing:
 - Overlord
 - Middle Manager

19.2 Also

- Deep Storage (S3, HDFS, ...)
- Metadata Storage (SQL)
- Load Balancer
- Cache

19.3 Coordinator

- Real-time Nodes (pull data, index it)
- Historical Nodes (keep old segments)
- Broker Nodes (route queries to RT & Hist. nodes, merge)
- Coordinator (manage segments)
- For indexing:
 - Overlord (distribute task to the middle manager)
 - Middle Manager (execute tasks via Peons)

20 When *not* to choose Druid

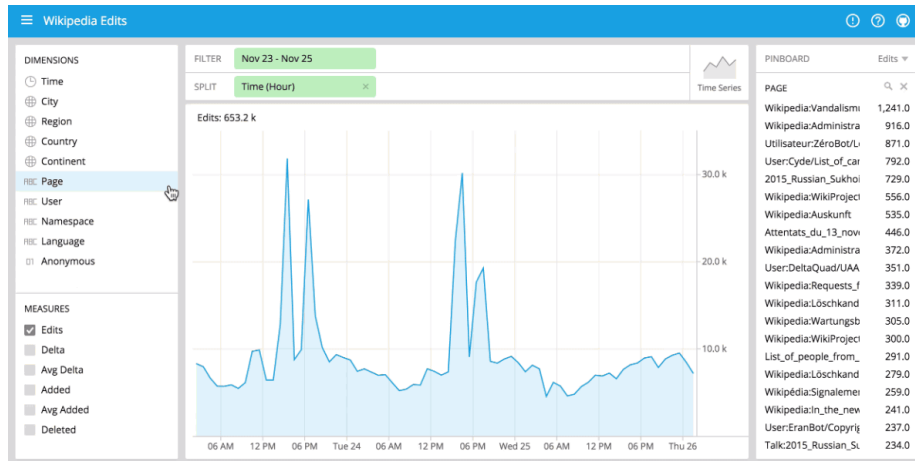
- Data is not time-series
- Cardinality is *very* high
- Number of dimensions is high
- Setup cost must be avoided

21 Graphite (metrics)



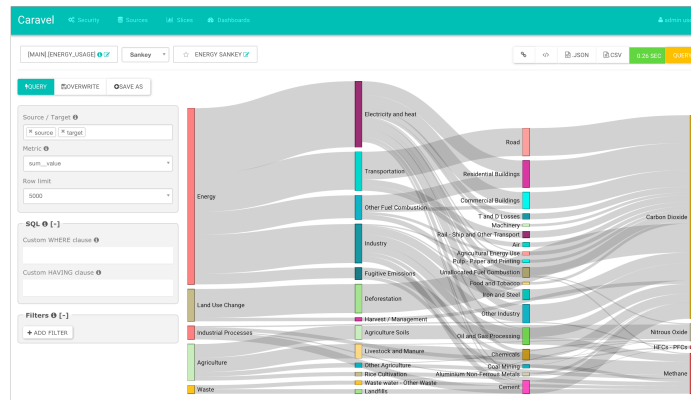
Graphite

22 Pivot (exploring data)



Pivot

23 Caravel



Caravel

24 Conclusions

24.1 Precompute your time series?



24.2 Don't reinvent it

- need a user facing API
- need time series on many dimensions
- need real-time
- big volume of data

24.3 Druid way is the right way!

1. Push in kafka
2. Add the right dimensions
3. Push in druid
4. ???
5. Profit!