Querying 100TB w/BigQuery and Spark

Jon Chase
jon@cedexis.com
@jonchase

Agenda

- Overview
- BigQuery
- Spark

Overview

cedexis

Closed-loop automatic optimization at web scale

cedexis

Our Volume

- 7.3 billion transactions/day
  - 84.7K/sec (average)
  - peak TPS is higher

Retention

- Logs: 1 month
- Minute aggregates: 1 month*
- Hour aggregates: 2 months*
- Day aggregates: 13 months*

- 120TB in BigQuery and growing

* Aggregations are loss-less

cedexis

Our use cases

- User facing charts in our portal (interactive)
  - Heavy use of percentiles/histograms
- Ad hoc analysis by our engineers and SEs (and more!) – “data driven”
- Complex “show me the value” jobs

What we wanted

- Unlimited storage
- Lossless (or not very lossy)
- Fast
- Low maintenance
- Easy to use
- Monitor-able
- Cost effective
- Horizontally scalable

BigQuery

BigQuery overview

- “A fast, economical and fully managed data warehouse for large-scale data analytics.”

BigQuery overview

- SQL access for “big data”
- SaaS
- Pay as you go
- Excellent docs
- Columnar
- Horizontally scales
- Much faster than Hive/Pig/Spark SQL, etc.
- Cheaper/easier/more powerful than Impala

Our BQ use cases

- Querying raw logs
  - Typically done by engineers when debugging
- Querying aggregations
  - Preaggregate data for interactive web charts

Loading data into BQ

- Formats: JSON, CSV

- Batch (cheaper, slower, 1-2 min load delay)
- Stream ($, faster, instantly query-able)

Organizing data

- Partitions
  - By day, by customer, preferably both
- Querying partitions
  - `table_query(my_dataset, 'table_id contains "20160201"')`
  - …or…
  - `from table1, table2, table3`

cedexis

Schemas


"timestamp", "type": "TIMESTAMP", "mode": "REQ
"clientMarketId", "type": "INTEGER", "mode": "
"clientCountryId", "type": "INTEGER", "mode":
"clientRegionId", "type": "INTEGER", "mode": "
"clientStateId", "type": "INTEGER", "mode": "R
"clientAsnId", "type": "INTEGER", "mode": "REQ
"reporterZoneId", "type": "INTEGER", "mode": "
"reporterCustomerId", "type": "INTEGER", "mode
"resolverMarketId", "type": "INTEGER", "mode":

Data types

- Integer
- Float
- String
- Timestamp
- Boolean
- Nested
- Repeated

Nested/repeated example

pageViewId	pageViewReportOrder	rt.protocol	rt.host	rt.path
Z3V1JndmRIQjNUNTZTODluajdZRGc2dmZwZ3hkRg--	1	http	yzutbfns.com	/acttr?p=YTM1MDQxNjAwNjNUnc2LwlXz%2FJLmtsxfgJYFvBqeqTpxKCc52ZMoeBpE9LwAQZepK5H6n6ZO
		http	rpt.cedexis.com	/f1/_CgJqMRASGBUiBggBEI-aASiK4-rNDTCd5OW1BTio5OW1BUDRy46sBkoUCAEQ3wEY7z4gh4CAwAQ
		http	rpt.cedexis.com	/f1/_CgJqMRASGBUiBggBEI-aASiK4-rNDTCd5OW1BTio5OW1BUDRy46sBkoUCAEQ3wEY7z4gh4CAwAQ
		http	rpt.cedexis.com	/f1/_CgJqMRASGBUiBggBEI-aASiK4-rNDTCd5OW1BTio5OW1BUDRy46sBkoUCAEQ3wEY7z4gh4CAwAQ
		http	rpt.cedexis.com	/f1/_CgJqMRASGBUiBggBEI-aASiK4-rNDTCd5OW1BTio5OW1BUDRy46sBkoUCAEQ3wEY7z4gh4CAwAQ
		http	rpt.cedexis.com	/n1/0/1454993944869/0/0/0/0/1454993945399/1454993945399/1454993945399/1454993945399/145499394
		http	rpt.cedexis.com	/r1/1/19727/_CgJqMRASGBUiBggBEI-aASiK4-rNDTCd5OW1BTio5OW1BUDRy46sBkoUCAEQ3wEY7z4gh4
		http	radar.cedexis.com	/1/19727/radar/1454527693/HPJ8p8zHLn7hpq3Ls5zC/providers.json?imagesok=1&r=1&t=1&p=1&n=1&l=0&
		http	radar.cedexis.com	/1454527693/radar/main.js?a=1&b=2&l=0&n=1&p=1&t=1&r=1&imagesok=1
		http	prod.content.animaliam.com	/1261/imageArrays/5bf673ce74d7860472377d7eb6f5f7bb?v=1

cedexis

Querying

- Mostly just ‘regular’ SQL
- Full complement of SQL is available
- Plus BQ specific additions

cedexis

```
SELECT
  group_0_dimension,
  group_1_dimension,
  STRFTIME_UTC_USEC(group_2_dimension, "%FT%TZ") group_2_dimension,
  group_2_fact_0
FROM (
  SELECT
    group_0_dimension,
    group_1_dimension,
    group_2_dimension,
    group_2_fact_0,
    DENSE_RANK() OVER (ORDER BY group_0_sort DESC) group_0_rank,
    DENSE_RANK() OVER (PARTITION BY group_0_dimension ORDER BY group_1_sort DESC) group_1_rank
  FROM (
    SELECT
      group_0_dimension,
      group_1_dimension,
      group_2_dimension,
      SUM(total) OVER (PARTITION BY group_0_dimension, group_1_dimension, group_2_dimension) group_2_fact_0,
      SUM(total) OVER (PARTITION BY group_0_dimension) group_0_sort,
      SUM(total) OVER (PARTITION BY group_0_dimension, group_1_dimension) group_1_sort,
      group_2_dimension group_2_sort
    FROM (
      SELECT
        appId group_0_dimension,
        effectiveCountryId group_1_dimension,
        [timestamp] group_2_dimension,
```

cedexis

```
 SUM(total) total
      FROM
        [openmix_minute_zid_cid.1_13949_20160221_19],
        [openmix_minute_zid_cid.1_13949_20160222_14],
        [openmix_minute_zid_cid.1_13949_20160222_16]
      WHERE
        [timestamp] BETWEEN TIMESTAMP("2016-02-21 16:00:00 UTC")
        AND TIMESTAMP("2016-02-22 16:00:00 UTC")
        AND appId IN (11)
      GROUP BY
        group_0_dimension,
        group_1_dimension,
        group_2_dimension ) )
  ORDER BY
    group_0_sort DESC,
    group_1_sort DESC,
    group_2_sort ASC )
WHERE
  group_0_rank <= 1
  AND group_1_rank <= 5
```

cedexis

```
SELECT COUNT(1) AS COUNT,
       clientIp
FROM TABLE_QUERY(radar_log, 'table_id contains "20160204"')
WHERE PARSE_IP(clientIp) IS NULL
  AND NOT clientIp CONTAINS ":"
GROUP BY 2
ORDER BY 1 DESC
```

Valid: This query will process 69.3 GB when run.

RUN QUERY | Save Query | Save View | Format Query | Show Options

Query complete (41.5s elapsed, 69.3 GB processed)

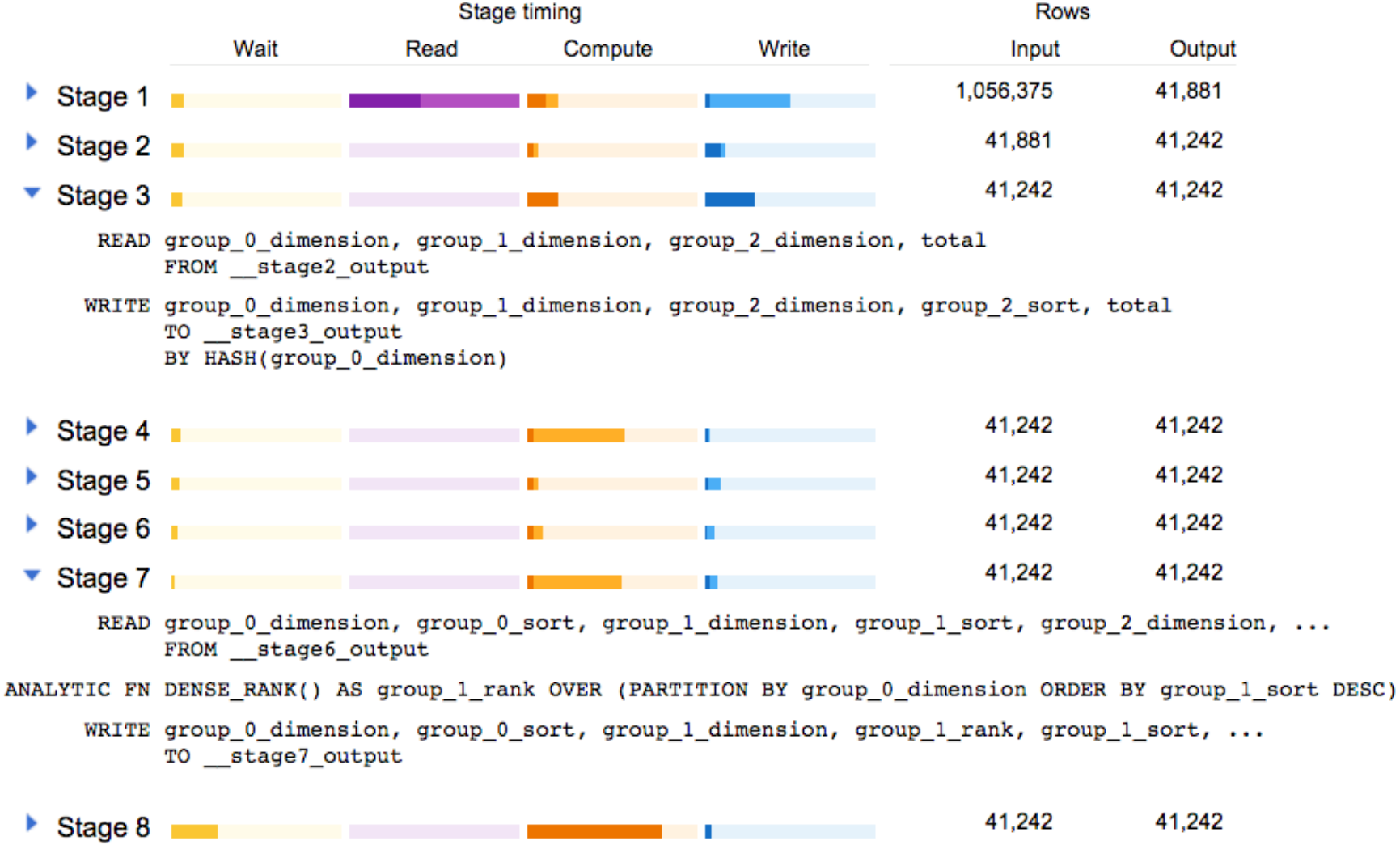
Query Results

Feb 5, 2016, 2:41:02 PM

Timeseries Graph | Column XY Graph | Download as CSV | Download as JSON | Save a

Row	COUNT	clientIp
1	18	unknown

cedexis


Stage timing
Wait
Read
Compute
Write
Rows
Input
Output
Stage 1
1,056,375
41,881
Stage 2
41,881
41,242
Stage 3
41,242
41,242
READ group_0_dimension, group_1_dimension, group_2_dimension, total
FROM __stage2_output
WRITE group_0_dimension, group_1_dimension, group_2_dimension, group_2_sort, total
TO __stage3_output
BY HASH(group_0_dimension)
Stage 4
41,242
41,242
Stage 5
41,242
41,242
Stage 6
41,242
41,242
Stage 7
41,242
41,242
READ group_0_dimension, group_0_sort, group_1_dimension, group_1_sort, group_2_dimension, ...
FROM __stage6_output
ANALYTIC FN DENSE_RANK() AS group_1_rank OVER (PARTITION BY group_0_dimension ORDER BY group_1_sort DESC)
WRITE group_0_dimension, group_0_sort, group_1_dimension, group_1_rank, group_1_sort, ...
TO __stage7_output
Stage 8
41,242
41,242

Query performance

- Quite good in general, though…
- Best case is never as good as RDBMS, but…
- Worst case is much better, however…
- Shared resources, so performance will vary

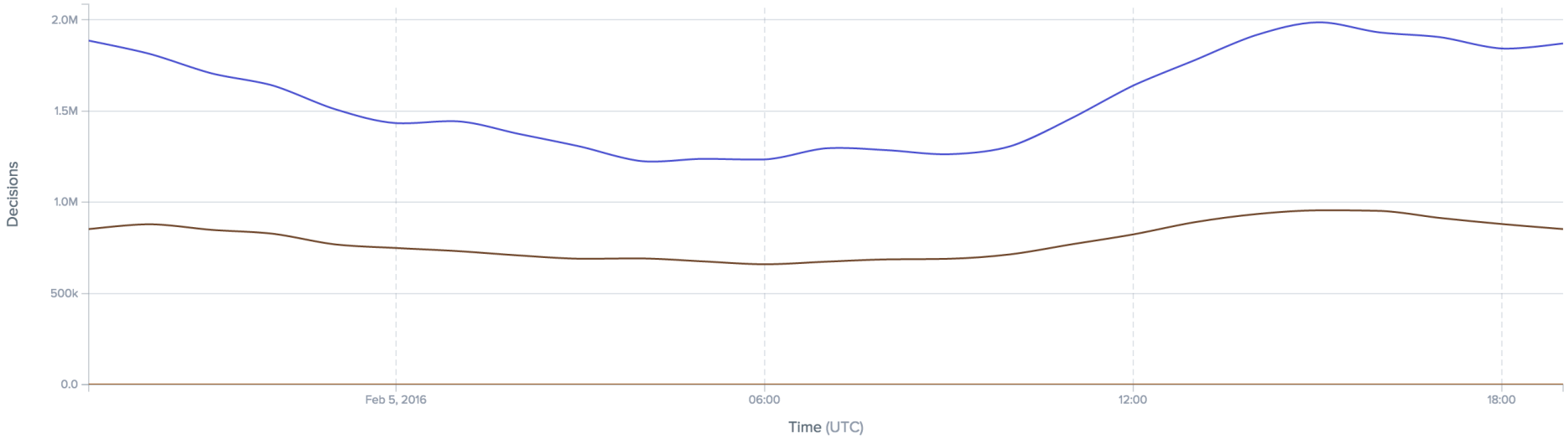
Storage space and retention

- Unlimited (in practice)
- Simply a function of how much you want to spend

Decision Report

USE BIGQUERY | APPLICATION | NONE | LAST 24 HOURS

Filters: Last 24 Hours, Only My Visitors, Decisions

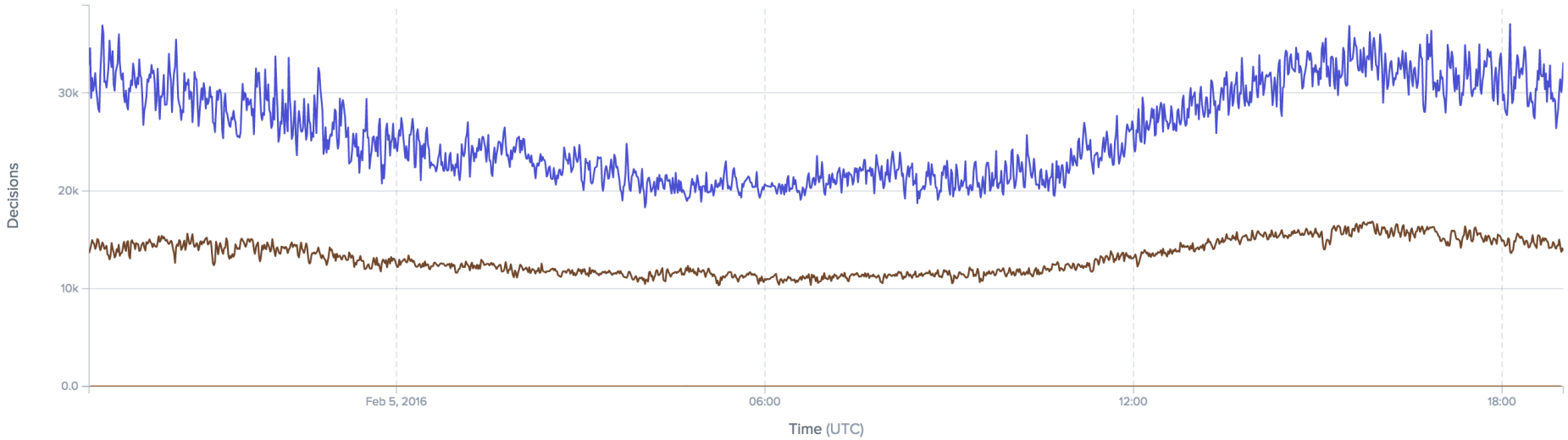


cedexis

Decision Report

USE BIGQUERY | APPLICATION | NONE | LAST 24 HOURS

Filters: Last 24 Hours, Only My Visitors, Decisions

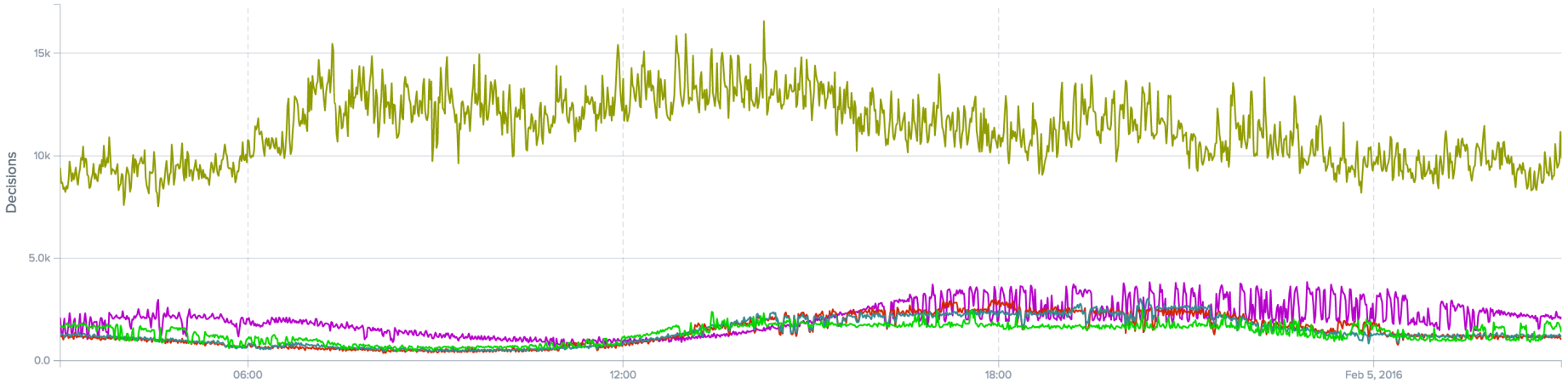


cedexis

Decision Report

USE BIGQUERY | APPLICATION | REGION | LAST 24 HOURS

Filters: Last 24 Hours, Only My Visitors, Decisions



cedexis

Aging out old data

- Set expiration time on table when you create it
- Set default expiration time per dataset, e.g.
  - Minute: 1 month
  - Hour: 2 months
  - Day: 13 months
- BQ cleans up after you

Tuning (hardware/settings/etc.)

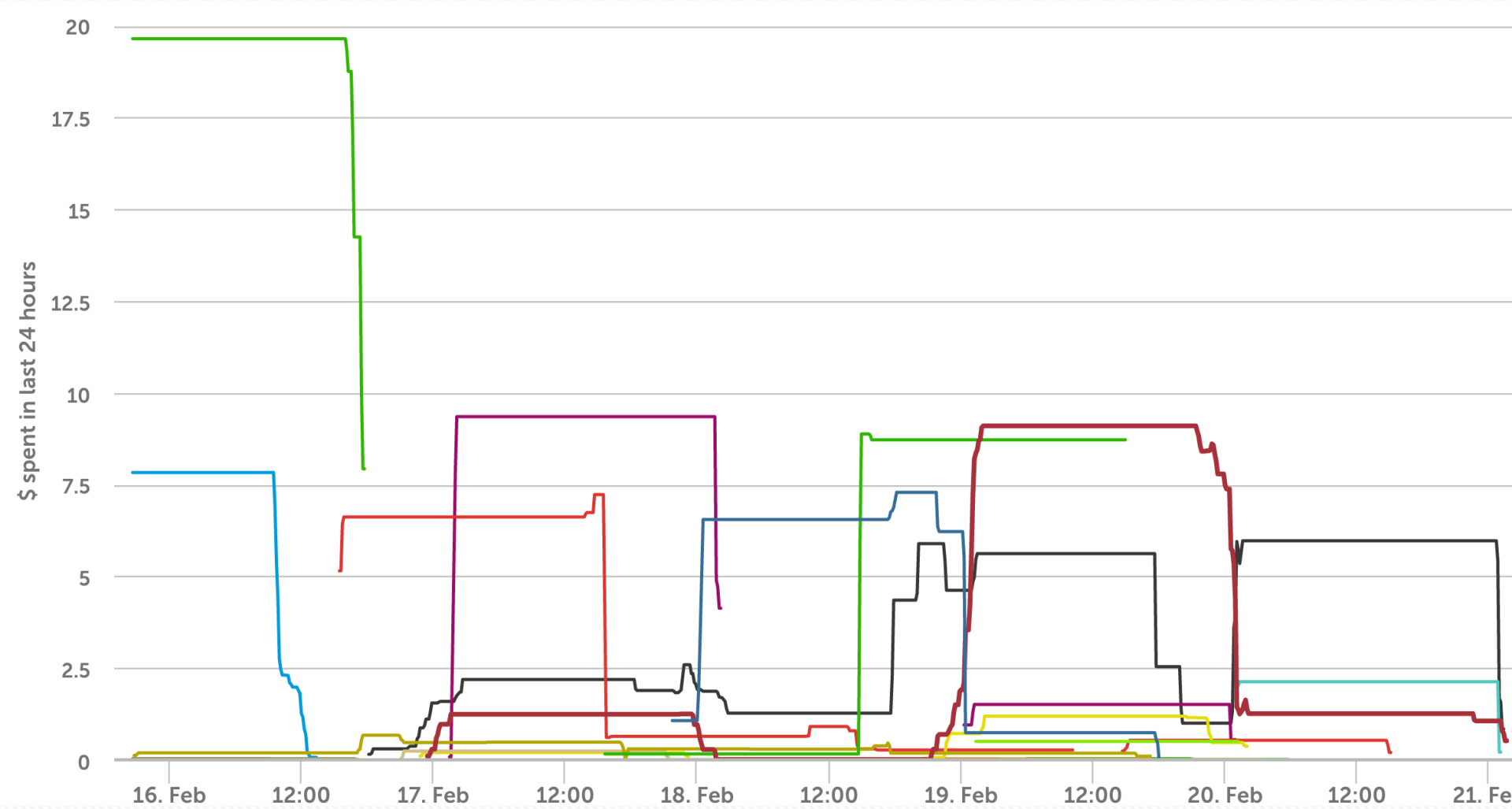
<this slide intentionally left blank>

BigQuery API

- All major languages
- Quite nice to work with

Monitoring

- API is helpful to monitor load jobs and queries
- Table metadata (size, row count, creation time, etc.) available via SQL


20
17.5
15
12.5
10
7.5
5
2.5
0
$ spent in last 24 hours
16. Feb
12:00
17. Feb
12:00
18. Feb
12:00
19. Feb
12:00
20. Feb
12:00
21. Fe

Cost

- Store: $0.02/GB/mo.
  - (100TB = $2,048/mo.)
- Query: $5.00/TB
- Streaming inserts: $0.01/200MB
- Batch inserts: $0

When **not** to use BQ

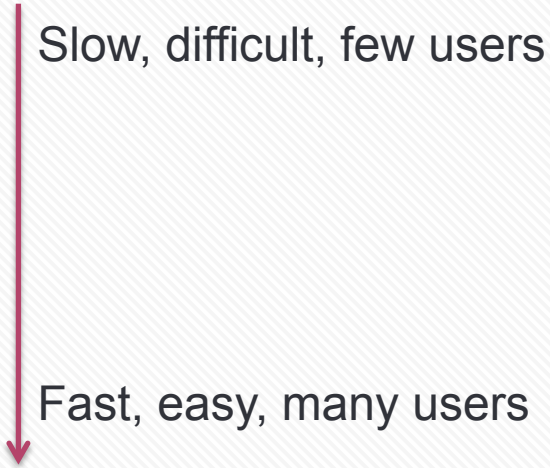
- …if all you want is the absolute fastest query performance…it might not be for you
- …if you can pre-aggregate everything
- …if you can fit it into an RDBMS or KV store

When **to** use BigQuery

- However, if you want to empower anyone in your organization to dig into the data, it may be worth it
- You can always mix and match BQ (data exploration/debugging) with another backend solution (snappy charts)

Evolution of data access @ Cedexis

- Shell scripts
- Hadoop jobs
- Spark SQL
- BigQuery



BigQuery - conclusion

- BigQuery = good

Spark

Spark

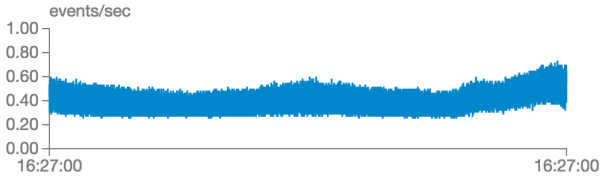
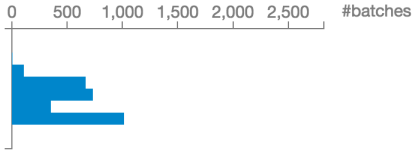
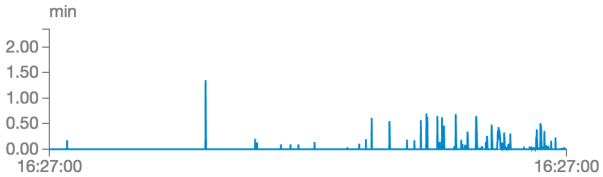
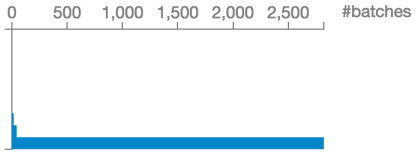
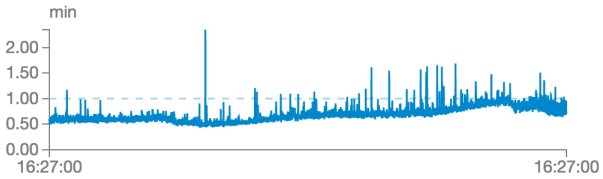
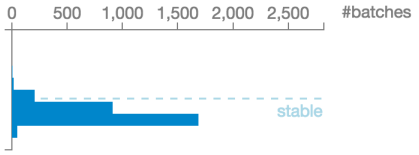
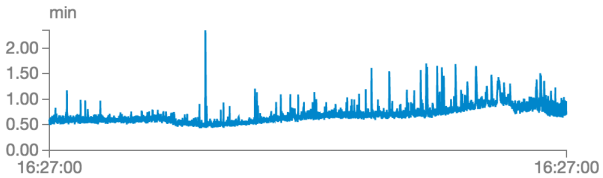
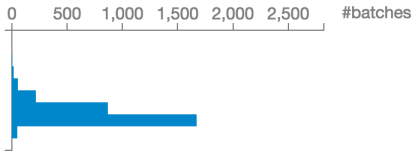
- Distributed processing framework on the JVM
- Like Hadoop, but not
- Execute arbitrary code in distributed fashion
- Horizontally scalable

Spark Streaming

- Runs in batches
- 24/7
- ETLs logs into BigQuery
- Reuse code between raw logs and aggregates

Streaming Statistics

Running batches of **1 minute** for **2 days 19 hours 10 minutes** since **2016/02/19 21:17:40** (**4029** completed batches, **96980** records)

	Timelines (Last 2881 batches, 1 active, 2880 completed)	Histograms
Input Rate Avg: 0.40 events/sec		
Scheduling Delay (?) Avg: 747 ms		
Processing Time (?) Avg: 41 seconds 872 ms		
Total Delay (?) Avg: 42 seconds 620 ms		

cedexis

Active Batches (1)

Batch Time	Input Size	Scheduling Delay (?)	Processing Time (?)	Status
2016/02/22 16:27:00	39 events	0 ms	-	processing

Completed Batches (last 2880 out of 4029)

Batch Time	Input Size	Scheduling Delay (?)	Processing Time (?)	Total Delay (?)
2016/02/22 16:26:00	23 events	0 ms	42 s	42 s
2016/02/22 16:25:00	42 events	0 ms	58 s	58 s
2016/02/22 16:24:00	28 events	0 ms	44 s	44 s
2016/02/22 16:23:00	33 events	0 ms	46 s	46 s
2016/02/22 16:22:00	25 events	0 ms	41 s	41 s
2016/02/22 16:21:00	40 events	0 ms	58 s	58 s
2016/02/22 16:20:00	19 events	0 ms	41 s	41 s
2016/02/22 16:19:00	42 events	0 ms	59 s	59 s
2016/02/22 16:18:00	26 events	0 ms	47 s	47 s
2016/02/22 16:17:00	40 events	1 ms	58 s	58 s
2016/02/22 16:16:00	21 events	0 ms	41 s	41 s
2016/02/22 16:15:00	39 events	0 ms	54 s	54 s

cedexis

Data flow

- Logs -> S3 -> SQS -> Spark Streaming -> BigQuery
- Happens once per minute

Spark - conclusion

- Spark = good

Q/A

cedexis