API Analytics with Redis and Google BigQuery
moral of the story

you can do big,
if you know how
API Analytics with Redis and Google BigQuery

javier ramirez
@supercoco9
REST API
+
AngularJS web as an API client
How many visits... did you say you have?
obvious solution:
use a ready-made service
(3scale, apigee...)

SMELLS LIKE BIG DATA
data that’s an order of magnitude greater than data you’re accustomed to

Doug Laney
VP Research, Business Analytics and Performance Management at Gartner
data that exceeds the processing capacity of conventional database systems. The data is too big, moves too fast, or doesn’t fit the structures of your database architectures.

Ed Dumbill

program chair for the O’Reilly Strata Conference
bigdata is doing a fullscan to 330MM rows, matching them against a regexp, and getting the result (223MM rows) in just 5 seconds

Javier Ramirez
impresionable teowaki founder
1. non intrusive metrics
2. keep the history
3. avoid vendor lock-in
4. interactive queries
5. cheap
6. extra ball: real time
MEASUREMENTS SHOULDN'T AFFECT OUR SYSTEM
Intel(R) Xeon(R) CPU E5520 @ 2.27GHz (with pipelining)

$ ./redis-benchmark -r 1000000 -n 2000000 -t get,set,lpush,lpop -P 16 -q

SET: 552,028 requests per second  
GET: 707,463 requests per second  
LPUSH: 767,459 requests per second  
LPOP: 770,119 requests per second

Intel(R) Xeon(R) CPU E5520 @ 2.27GHz (without pipelining)

$ ./redis-benchmark -r 1000000 -n 2000000 -t get,set,lpush,lpop -q

SET: 122,556 requests per second  
GET: 123,601 requests per second  
LPUSH: 136,752 requests per second  
LPOP: 132,424 requests per second
open source, BSD licensed, advanced key-value store. It is often referred to as a data structure server since keys can contain strings, hashes, lists, sets, sorted sets and hyperloglogs.

http://redis.io

started in 2009 by Salvatore Sanfilippo @antirez

111 contributors at
https://github.com/antirez/redis
what is it used for?
twitter

Every time line (800 tweets per user) is stored in redis

5000 writes per second avg

300K reads per second
youporn

Most data is found in hashes with ordered sets used to know what data to show.

```
```

Then perform a zRange to get the pages we want and get the list of video_ids back.

Then start a pipeline and get all the videos from hashes.
I USE REDIS FOR EVERYTHING

HIPSTER
Redis keeps everything in memory all the time.
WHERE ARE YOU GOING TO STORE

ALL THOSE DATA
easy:
store GZIPPED files into S3/Glacier

* or google cloud storage
ARE YOU GOING TO QUERY.. 

MILLIONS OF ROWS INTERACTIVELY?
Hadoop (map/reduce)

http://hadoop.apache.org/

started in 2005 by Doug Cutting and Mike Cafarella
Cassandra

http://cassandra.apache.org/

released in 2008 by facebook.
other big data solutions:

Hadoop+Voldemort+Kafka

http://engineering.linkedin.com/projects

Hbase

http://hbase.apache.org/
Amazon Redshift

http://aws.amazon.com/redshift/
Our choice:

Google BigQuery

Data analysis as a service

http://developers.google.com/bigquery
Based on Dremel

Specifically designed for interactive queries over petabytes of real-time data
Columnar storage

Easy to compress

Convenient for querying long series over a single column
loading data

You can feed flat CSV-like files or nested JSON objects
bq load --nosynchronous_mode
--encoding UTF-8
--field_delimiter 'tab'
--max_bad_records 100
--source_format CSV
api.stats
20131014T11-42-05Z.gz
web console screenshot

```
5_most_created_resources

1 select count(*) from api_dev.stats
```

Query Results

<table>
<thead>
<tr>
<th>Row</th>
<th>f0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>382087</td>
</tr>
</tbody>
</table>
almost SQL

select    avg    &    =    AND
from      count  |    !=    OR
join      max    ^    <>    NOT
where     min    <<   >
group by  sum    >>   <
having    ~     >=   <=
order     +     IN
limit     -     IS NULL
           *     BETWEEN
           /     
           %
## Functions overview

<table>
<thead>
<tr>
<th>Date and Time Functions</th>
<th>Mathematical Functions</th>
<th>String Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>current_date</td>
<td>abs</td>
<td>concat</td>
</tr>
<tr>
<td>current_time</td>
<td>acos</td>
<td>contains</td>
</tr>
<tr>
<td>now</td>
<td>atan</td>
<td>left</td>
</tr>
<tr>
<td>datediff</td>
<td>ceil</td>
<td>length</td>
</tr>
<tr>
<td>day</td>
<td>floor</td>
<td>lower</td>
</tr>
<tr>
<td>day_of_week</td>
<td>degrees</td>
<td>upper</td>
</tr>
<tr>
<td>day_of_year</td>
<td>log</td>
<td>lpad</td>
</tr>
<tr>
<td>hour</td>
<td>log2</td>
<td>rpad</td>
</tr>
<tr>
<td>minute</td>
<td>log10</td>
<td>right</td>
</tr>
<tr>
<td>quarter</td>
<td>PI</td>
<td>substr</td>
</tr>
<tr>
<td>year...</td>
<td>SQRT...</td>
<td></td>
</tr>
</tbody>
</table>
specific extensions for analytics

within flatten nest

stddev

top first last

nth

variance

var_pop var_samp

covar_pop covar_samp

quantiles
window functions

```
select country, uri, total, ratio to report(total)
OVER (PARTITION BY country order by total DESC) ratio from
(select country, uri, count(*) total, from api_dev.stats
where method = 'POST' and country in ('DE', 'ES')
group by country, uri)
```
correlations.

not to mistake with causality
views.

JSON fields.

timestamped tables.
Things you always wanted to try but were too scared to

select count(*) from publicdata:samples.wikipedia

where REGEXP_MATCH(title, "[0-9]*")
AND wp_namespace = 0;

223,163,387
Query complete (5.6s elapsed, 9.13 GB processed, Cost: 32¢)
SELECT repository_name, repository_language, repository_description, COUNT(repository_name) as cnt, repository_url 
FROM github.timeline 
WHERE type="WatchEvent" 
AND PARSE_UTC_USEC(created_at) >= PARSE_UTC_USEC("#{yesterday} 20:00:00") 
AND repository_url IN ( 
  SELECT repository_url 
  FROM github.timeline 
  WHERE type="CreateEvent" 
  AND PARSE_UTC_USEC(repository_created_at) >= PARSE_UTC_USEC('#{yesterday} 20:00:00') 
  AND repository_fork = "false" 
  AND payload_ref_type = "repository" 
  GROUP BY repository_url 
) 
GROUP BY repository_name, repository_language, repository_description, repository_url 
HAVING cnt >= 5 
ORDER BY cnt DESC 
LIMIT 25
country segmented traffic

```
1. select country, count(*) total from api_dev.stats
2. group by country order by total DESC

Query complete (1.7s elapsed, 1.45 MB processed, Cost: 0$)
```

<table>
<thead>
<tr>
<th>Row</th>
<th>country</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GB</td>
<td>110901</td>
</tr>
<tr>
<td>2</td>
<td>US</td>
<td>108940</td>
</tr>
<tr>
<td>3</td>
<td>ES</td>
<td>106314</td>
</tr>
<tr>
<td>4</td>
<td>DE</td>
<td>55622</td>
</tr>
<tr>
<td>5</td>
<td>null</td>
<td>510</td>
</tr>
</tbody>
</table>
our most active user

```sql
select guid, count(*) total from api_dev.stats
  group by guid order by total DESC LIMIT 1;
```

**Query Results** 7:34pm, 17 Oct 2013

<table>
<thead>
<tr>
<th>Row</th>
<th>guid</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>105</td>
<td>382069</td>
</tr>
</tbody>
</table>
10 request we should be caching

```
10_most Likely_cacheable_actions

1 select top(uri, 10), count(*) total from api_dev.stats
2 where method = 'GET' and querystring IS NULL
3
```

**Query Results** 11:27pm, 17 Oct 2013

<table>
<thead>
<tr>
<th>Row</th>
<th>f0_</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>/users/tw/notifications/count</td>
<td>84011</td>
</tr>
<tr>
<td>2</td>
<td>/users/tw/links</td>
<td>31383</td>
</tr>
<tr>
<td>3</td>
<td>/users/tw/admin-orgs</td>
<td>31379</td>
</tr>
<tr>
<td>4</td>
<td>/users/tw/bp-categories</td>
<td>31369</td>
</tr>
<tr>
<td>5</td>
<td>/users/tw/stream</td>
<td>26151</td>
</tr>
<tr>
<td>6</td>
<td>/users/tw/practices</td>
<td>26142</td>
</tr>
<tr>
<td>7</td>
<td>/users/tw/link-categories</td>
<td>20922</td>
</tr>
<tr>
<td>8</td>
<td>/users/tw/shouts</td>
<td>15681</td>
</tr>
<tr>
<td>9</td>
<td>/teams/433-equipillo/bp-categories</td>
<td>10454</td>
</tr>
<tr>
<td>10</td>
<td>/users/tw/notifications</td>
<td>5227</td>
</tr>
</tbody>
</table>
5 most created resources

select uri, count(*) total from stats where method = 'POST' group by URI;
...but

/users/javier/shouts
/users/rgo/shouts
/teams/javier-community/links
/teams/nosqlmatters-cgn/links
5 most created resources

```
select REGEXP_EXTRACT(uri, r'.+(/.+)/z') resource, count(*) total from api_dev.stats
where method = 'POST'
group by resource order by total DESC limit 5;
```
Automation with Apps Script

Read from bigquery

Create a spreadsheet on Drive

E-mail it everyday as a PDF
Do you remember the future?
redis pricing

2* machines (master/slave) at digital ocean

$10 monthly

* we were already using these instances for a lot of redis use cases
s3 pricing

$0.095 per GB

a gzipped 1.6 MB file stores 300K rows

$0.0001541698 / monthly
glacier pricing

$0.01 per GB

$0.000016 / monthly
bigquery pricing

$80 per stored TB
300 000 000 rows => $0.007629392 / month

$35 per processed TB
1 full scan = 84 MB
1 count = 0 MB
1 full scan over 1 column = 5.4 MB
10 GB => $0.35 / month
redis $10.0000000000
s3 storage $00.0001541698
s3 transfer $00.0050000000
glacier transfer $00.0500000000
glacier storage $00.0000160000
bigquery storage $00.0076293920
bigquery queries $00.3500000000

$10.41 / month
1. non intrusive metrics
2. keep the history
3. avoid vendor lock-in
4. interactive queries
5. cheap
6. extra ball: real time
Danke!

Javier Ramírez
@supercoco9

Find related links at
https://teowaki.com/teams/javier-community/link-categories/bigquery-talk