Big Data

Facebook Wall Data using Graph API

Presented by:
Prashant Patel-2556219
Jaykrushna Patel-2619715
Outline

• Data Source
• Processing tools for processing our data
• Big Data Processing System: Mongodb
• Data Transformation-Mongodb
• Data Mining Queries
• References
• Research Paper
Data Source

• Downloaded from Facebook Graph API
• Approach : Python Script
  • FQL
    • Facebook Query Language
  • Access Token
    • We received using authorize through our own Application
```python
import requests
import json
import sys

after = '1353233754'
token = 'CAACEDeoseoCBABolXtGsDSdsmNioQ16uujuvfiBKJ4lUhx8BZb2YGZAtHup30dk1gqzUHwd4lz5sk0bZAZdZBBZClby9l8SywnC85pznZAIva11slqT9r2MlMNkWCjDjZCOS

def get_posts():
    """Returns dictionary of id, first names of people who posted on my wall
    between start and end time"""
    query = ('SELECT post_id, actor_id, message, created_time, likes,comments FROM stream WHERE "filter_key = 'others' LIMIT 250000"
    payload = {'q': query, 'access_token': token}
    r = requests.get('https://graph.facebook.com/fql', params=payload)
    result = json.loads(r.text)

    name = '1.txt'  # Name of text file coerced with .txt

    try:
        with open(name, 'w') as outfile:
            json.dump(result, outfile)
    except:
        print('Something went wrong! Can\'t tell what?')
        sys.exit(0)  # quit Python

    """print result""

if __name__ == '__main__':
    get_posts()
```
Sample data

```json
{
  "comments": [
    {
      "count": 1,
      "comment_list": [
        {
          "text": "v three v proving ourselves... N this is the only our lifetime achievement",
          "post_fid": "788356244539086",
          "text_tags": [],
          "likes": 0,
          "time": 1416038370,
          "fromid": "100000881525068",
          "id": "100000881525068_788355811205796_788356244539086",
          "user_likes": false
        }
      ],
      "can_remove": false,
      "can_post": true
    },
    {
      "post_id": "100000881525068_788355811205796",
      "actor_id": "100000881525068",
      "created_time": 1416033673,
      "message": "Missing Childhood... Whatever it was but actually very very beautiful n awesome whenever u miss...Love U my "Ben" this is the word which I st"
    }
  ],
  "can_remove": true,
  "can_like": true,
  "sample": [
    100003693026000,
    100001785256196,
    100002821210599
  ],
  "href": "https://www.facebook.com/browse/likes/?id=788335811205796",
  "friends": [
    100000881525068
  ],
}
```
Possible tools for processing our data

- Spark
- Hive
- MongoDB
Spark-Easy Json data manipulation

- The entry point into all relational functionality in Spark is the SQLContext class. So, here we create SQLContext for querying sql command.

```
Welcome to  
Spark  
version 1.0.2

Using Scala version 2.10.4 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_25)
Type in expressions to have them evaluated.
Type :help for more information.
Spark context available as sc.

scala> val sqlContext = new org.apache.spark.sql.SQLContext(sc)
sqlContext: org.apache.spark.sql.SQLContext = org.apache.spark.sql.SQLContext@6249a08d
```

Here, we load a JSON file (one object per line), returning the result as a SchemaRDD.
Schema of the data

```scala
scala> val jsonData1 = sqlContext.jsonFile("/home/jaykrushna/Downloads/1.json")

scala> jsonData1.printSchema
root
 |-- data: ArrayType[org.apache.spark.sql.catalyst.types.StructType$1a4cbcc6]
|    |-- actor_id: LongType
|    |-- comments: org.apache.spark.sql.catalyst.types.StructType$1a4cbcc6
|    |    |-- can_post: BooleanType
|    |    |-- can_remove: BooleanType
|    |    |-- comment_list: ArrayType[org.apache.spark.sql.catalyst.types.StructType$1a4cbcc6]
|    |    |    |-- fromid: LongType
|    |    |    |-- id: StringType
|    |    |    |-- likes: IntegerType
|    |    |    |-- post_fbid: StringType
|    |    |    |-- text: StringType
|    |    |    |-- text_tags: ArrayType[org.apache.spark.sql.catalyst.types.StructType$1a4cbcc6]
|    |    |    |    |-- id: LongType
|    |    |    |    |-- length: IntegerType
|    |    |    |    |-- name: StringType
|    |    |    |    |-- offset: IntegerType
|    |    |    |    |-- type: StringType
|    |    |    |    |-- time: IntegerType
|    |    |    |    |-- user_likes: BooleanType
|    |    |    |    |-- count: IntegerType
|    |    |    |-- created_time: IntegerType
|    |    |    |-- likes: org.apache.spark.sql.catalyst.types.StructType$1a4cbcc6
|    |    |    |    |-- can_like: BooleanType
|    |    |    |    |-- count: IntegerType
|    |    |    |    |-- friends: ArrayType[LongType]
|    |    |    |    |-- href: StringType
|    |    |    |    |-- sample: ArrayType[LongType]
|    |    |    |    |-- user_likes: BooleanType
|    |    |    |    |-- message: StringType
|    |    |    |    |-- post_id: StringType
```
• Create table in Spark. Here, we create temporary table jt1 for storing json Data.

• SQL statements can be run by using the sql methods provided by sqlContext. Here only display one row data from jt1 table.

```scala
cscala> jsonData1.registerAsTable("jt1")

cscala> sqlContext.sql("select data[4] from jt1 limit 1").collect.foreach(println)
[[653510123,[true,false,ArrayBuffer([100001627629997,653510123_10152390039620124_10152200091725124,1,10152200091725124,So cute ArrayBuffer(),1407611805,false], [100000427111249,653510123_10152390039620124_10152229942915124,1,10152229942915124,Marta serdecznie gratuluję, śliczna cór cia.,ArrayBuffer(),1408953111,false], [627827254,653510123_10152390039620124_10152287477265124,1,10152287477265124,Widzę na pierwszym zdjęciu, że trzymała wczeraj kciuki za Polaków.,ArrayBuffer(),1411395959,false], [100008172128658,653510123_10152390039620124_10152355095585124,0,10152355095585124,sliczna biedroneczka no i mama też <3,ArrayBuffer(),1414315659,false],47],1416001351,[true,203,ArrayBuffer(653510123),https://www.facebook.com/browse/likes/?id=10152045261660124,ArrayBuffer(634166766, 635705971, 801555342),false],653510123_10152390039620124]]

cscala>
```
Here, Display all data from jt1 table.
Why not Spark

• It does not support rich data mining queries.
• No support for interactive data mining Queries
Why not Hive:

• Hive is not able to process complex JSON Graph data
• No support for dynamic large schemas
Big Data Processing System: Mongodb

• MongoDB is an open-source document database, and the leading NoSQL database.

• Features:
  • Document-Oriented Storage
  • Full Index Support
  • Auto-Sharding
    • Scale horizontally without compromising functionality.
  • Querying
    • Rich, document-based queries.
  • Map/Reduce
    • Flexible aggregation and data processing.
  • GridFS
    • Store files of any size without complicating your stack.
• Mongodb Management service
  • Also support cloud services.
• Provide rich GUI Tools:
  • Edda
  • Fluentd
  • Fang of Mongo
  • Umongo
  • Mongo-vision
  • MongoHub
  • MongoVUE
  • RockMongo
  • Robomongo
• MongoDB belongs to the type of document-oriented database in which data is organized as JSON or BSON document, and store into an collection.

• Database holds a set of collections. collection holds a set of documents. Collection and document are same as table and row in relational database.

• Document is a set of key-value pairs. Documents have dynamic schema. Dynamic schema means that documents in the same collection do not need to have the same set of fields or structure, and common fields in a collection’s documents may hold different types of data.
Data Transformation-Mongodb

• Start Mongo Server
  • ./mongod

• Start Mongo Client
  • mongo
• Server

```
prashant@prashant-Lenovo-G510:~/Downloads/mongodb-linux-x86_64-2.6.5/bin$ ./mongod
./mongod --help for help and startup options
2014-11-23T16:00:25.241-0500 [initandlisten] MongoDB starting : pid=2575 port=27017 dbpath=/data/db
2014-11-23T16:00:25.241-0500 [initandlisten] db version v2.6.5
2014-11-23T16:00:25.241-0500 [initandlisten] git version: e99d4fcb4279c0279796f237aa92fe3b645
2014-11-23T16:00:25.241-0500 [initandlisten] build info: Linux build8.nj1.10gen.cc 2.6.32-43
2014-11-23T16:00:25.241-0500 [initandlisten] BOOST_LIB_VERSION=1_09
2014-11-23T16:00:25.241-0500 [initandlisten] allocator: tcmalloc
2014-11-23T16:00:25.241-0500 [initandlisten] options: {}
2014-11-23T16:00:25.346-0500 [initandlisten] journal dir=/data/db/journal
2014-11-23T16:00:25.346-0500 [initandlisten] recover : no journal files present, no recovery
2014-11-23T16:00:25.820-0500 [initandlisten] waiting for connections on port 27017
```

• Client

```
prashant@prashant-Lenovo-G510:~$ cd /home/prashant/Downloads/
prashant@prashant-Lenovo-G510:~$ cd /home/prashant/Downloads/
prashant@prashant-Lenovo-G510:~/Downloads$ cd mongodb-linux-x86_64-2.6.5
prashant@prashant-Lenovo-G510:~/Downloads/mongodb-linux-x86_64-2.6.5$ cd bin
prashant@prashant-Lenovo-G510:~/Downloads/mongodb-linux-x86_64-2.6.5$ cd bin
prashant@prashant-Lenovo-G510:~/Downloads/mongodb-linux-x86_64-2.6.5/bin$ mongo
MongoDB shell version: 2.4.9
connecting to: test
```
• Database

```
prashant@prashant-Lenovo-G510:~/Downloads/mongodb-linux-x86_64-2.6.5/bin$ mongo
MongoDB shell version: 2.4.9
connecting to: test
> show database
Sun Nov 23 16:01:54.494 don't know how to show [database] at src/mongo/shell/utils.js:847
> show databases
admin (empty)
big  0.078125GB
local 0.078125GB
test 0.078125GB
> use bigdata
switched to db bigdata
```

• Collections

```
> db.createCollection("graph", {capped: false, autoIndexId: true, size: 0, max: 0} )
{ "ok" : 1 }
> show collections
graph
system.indexes
```
• Import Data

mongoimport --db <db-name> --collection <coll-name> --type json --file seed.json --jsonArray
Imported data
Data Transformation

• Issues retrieving specific records
  • All Queries return Same records.
    • Data array of all elements.

• Reason:

  All data are in single array so its a single Object ID (Index).

• Solution
  • Transform the Array Elements into separate documents
    • Each of them having different INDEX.
• Approaches to Transform the data into individual documents.
  • Script (Python, Php)
  • Use Batch Insert functionality into MongoDB
• Before Applying transformation Strategy
  
• After Applying transformation Strategy
Data Mining Queries

• Retrieve all records from graph collection.
  
  `db.graph.find()`
Retrieve Records having Comments more than 0

db.graph.find({"comments.count" : {$gt : 0 }})
Return few fields instead the whole structure
Suppose we want only post_id and number of comments on it

```javascript
> db.graph.aggregate([{
  $project: {
    post_id: 1,
    comments: { $match: { count: 1 } },
    count: 1
  }
}])
```
null
Return 5 oldest message which is liked by at least one

```javascript
> db.graph.aggregate(
  ...
  [ { $sort: { created_time: 1 } },
  ...
  { $project:
    { message:1,
      "likes.count" : 1,
    }
  },
  ...
  { $limit : 5 }
  ...
  ]
```
... )

"result" : [

    {
        "id" : ObjectId("5472613e44ae878e1f8e132b"),
        "message" : "Actress Carol Ann Susi, best known as the voice of Howard Wolowitz’s unseen mother on The Big Bang Theory, has died at the age of 62 after a short battle with cancer. \nR.I.P Mrs. Wolowitz....You will be missed..",
        "likes" : {
            "count" : 4
        }
    },

    {
        "id" : ObjectId("547260994ae878e1f8e1322"),
        "message" : "English: Livin-in relationship\nHindi: बिने फेयर, हम तेरे \n-\n", 
        "likes" : {
            "count" : 12
        }
    },

    {
        "id" : ObjectId("5472618044ae878e1f8e132e"),
        "message" : "Avoiding something doesn’t mean you don’t want it. Sometimes, it means that you do want it, but you know it isn’t right for you.",
        "likes" : {
            "count" : 11
        }
    },

    {
        "id" : ObjectId("547261e944ae878e1f8e1333"),
        "message" : "Detroit Vs. Everybody (Audio)\nslimshady\nashebest\nshadyX\n",
        "likes" : {
            "count" : 1
        }
    },

    {
        "id" : ObjectId("5472621644ae878e1f8e1336"),
        "message" : "कुछ समयों पूरे करने में निकट हो गए है वह सभी गानों जो जादू है वह और मस्ती की है जा रही है. \n"n कभी नहीं होगी ये सुंदरता \n. \n"n हिंसक गुप्त \n. \n"n आप जारी रख देंगे \n. \n"
        "likes" : {
            "count" : 5
        }
    }
]
Return actor with total number of likes above 100 for all messages for each actor in descending order of total Likes

• Query

db.graph.aggregate([{
  _id: "$actor_id",
  totalLikes: {
    $sum: "$likes.count"
  }
},
{
  $sort: { totalLikes: -1 }
},
{
  $match: {totalLikes: { $gte : 100 } }
},
{
  $project:
  { _id: 0,
    actor: "$_id",
    Likes: "$totalLikes"
  }
}])

• Compare to SQL Query

```sql
SELECT actor_id AS actor, SUM(likes.count) AS Likes
FROM graph
GROUP BY actor_id
HAVING Likes >= (100)
ORDER BY Likes
```
Output:

```json
{
   "result":[
      {"actor":NumberLong("222401665690"),"Likes":311465},
      {"actor":NumberLong("216311481960"),"Likes":35940},
      {"actor":NumberLong("290897664260544"),"Likes":26998},
      {"actor":NumberLong("100000134387337"),"Likes":402},
      {"actor":653510123,"Likes":207},
      {"actor":1426470004,"Likes":177},
      {"actor":NumberLong("100001537437647"),"Likes":148},
      {"actor":NumberLong("100000484128282"),"Likes":144}
   ],
   "ok":1
}
```
Query return message and comment count and if there is no comments on the message then return “No Comments”

Query:

```javascript
> db.graph.aggregate(
...   [ 
...     { 
...       $project: 
...         { 
...           message: 1, 
...           comments: 
...             { 
...               $cond: { if: { $gt: [ "$comments.count", 0 ] }, then: "$comments.count", else: "No Comments" } 
...             } 
...         } 
...     } 
...   ]
...)
```
everyone should be proud of themselves."

"id" : ObjectId("5472613e44ae878e1f8e132b"),
"comments" : 9,
"message" : "Actress Carol Ann Susi, best known as the voice of Howard Wolowitz’s unseen mother on The Big Bang Theory, has died at the age of 62 after a short battle with cancer.\n\nI.P. Mrs. Wolowitz......You will be missed..."
},
{
"id" : ObjectId("547261344ae878e1f8e132c"),
"comments" : 584,
"message" : "Feeling McQueen all the way!"
},
{
"id" : ObjectId("547261544ae878e1f8e132d"),
"comments" : 499,
"message" : "Dr. Ada Igonoh survived Ebola in Lagos. Her moving story shows health workers at their best: http://b-gat.e/s/1100Gdd"
},
{
"id" : ObjectId("547261804ae878e1f8e132e"),
"comments" : "No Comments",
"message" : "Avoiding something doesn’t mean you don’t want it. Sometimes, it means that you do want it, but you know it isn’t right for you."
},
{
"id" : ObjectId("547261964ae878e1f8e132f"),
"comments" : "No Comments",
"message" : "#EBay Dumps #Google Syndicated Ads For Blng Ads On #Mobile_Devices...http://t.co/b0wP4OuMe"
},
{
"id" : ObjectId("547261a74ae878e1f8e1330"),
"comments" : "No Comments",
"message" : ""
},
{
"id" : ObjectId("547261b44ae878e1f8e1331"),
"comments" : "No Comments",
"message" : "Google Algorithm Chatter Heating Up, Shifts Happening?...https://t.co/oAXr0v4ca5"
Return selective fields inside nested structure
Return each post, messages, comments [count, comment_list [text]]

• Query

```javascript
> db.graph.aggregate([ { $project: { post_id: 1, comments: { count: 1, comment_list: { text: 1 } } } } ])
```
"comments" : {
  "count" : 8,
  "comment_list" : [ ]
},
"post_id" : "10000209084114_746818015478186"
},
{
  "_id" : ObjectId("5472622d4ae878e1f8e1337"),
  "comments" : {
    "count" : 1,
    "comment_list" : [
      {
        "text" : "Om nam sivay"
      }
    ]
  },
  "post_id" : "100009605233770_1558355177727975"
},
{
  "_id" : ObjectId("5472622d4ae878e1f8e1339"),
  "comments" : {
    "count" : 9,
    "comment_list" : [
      {
        "text" : "",
      },
      {
        "text" : "He is gone mad bro.....indian batsmen especially against Sri Lanka.."
      },
      {
        "text" : ""
      },
      {
        "text" : "Its good that it is SL not Pakistan........Otherwise People will celibate Diwali again.
        "
      }
    ]
  },
  "post_id" : "1000004842822_10105496022294501"
},

Return data with following Structure

```json
{
  "Biggestlikemessage": {
    "message": "test message",
    "TotalNoofLikes": NoofLikes(integer)
  },
  "Smallestlikemessage": {
    "message": "test message",
    "TotalNoofLikes": NoofLikes(integer)
  },
  "actor" (USER): actor_id(integer)
}
```
Query

```javascript
> db.graph.aggregate( {
  $group: {
    _id: { actor: "$actor_id", message: "$message" },
    Likes: { $sum: "$likes.count" } }
  },
  { $sort: { Likes: 1 } }
},
{ $group: {
  _id: "$_id.actor",
  biggestLikeMessage: { $last: "$_id.message" },
  maxTotalLikes: { $last: "$Likes" },
  smallestLikeMessage: { $first: "$_id.message" },
  leastTotalLikes: { $first: "$Likes" } }
},

// the following $project is optional, and
// modifies the output format.

{ $limit: 25 },
{ $project: {
  _id: 0,
  actor: "$_id",
  biggestLikeMessage: { message: "$biggestLikeMessage", TotalNoOfLikes: "$maxTotalLikes" },
  smallestLikeMessage: { message: "$smallestLikeMessage", TotalNoOfLikes: "$leastTotalLikes" }
} }
```
 temporarily unavailable.

"biggestLikeMessage": {
  "TotalNoofLikes": 3
},
"smallestLikeMessage": {
  "message": "Looking for Oneplus one? Now's the time to act! http://preorder.oneplus.net/",
  "TotalNoofLikes": 2
},
"actor": 660230888
},
{
  "biggestLikeMessage": {
    "message": "Fresh cut selfie. #brownsleek",
    "TotalNoofLikes": 70
  },
  "smallestLikeMessage": {
    "message": "The only thing I have to say about the weather is that taking the pup out for the first snowfall is always the BEST. Jessica D. Lewis I hope you and Homer had a fun first snow together!!",
    "TotalNoofLikes": 4
  },
  "actor": 1426470064
},
{
  "biggestLikeMessage": {
    "message": "#TakingOverTheWorld one step at a time..",
    "TotalNoofLikes": 129860
  },
  "smallestLikeMessage": {
    "message": "Feeling McQueen all the way!",
    "TotalNoofLikes": 41988
  },
  "actor": NumberLong("222401665900")
},
{
  "biggestLikeMessage": {
    "message": "सुकूम समय चलाकर निकल ने निकले बी रे रे

  "TotalNoofLikes": 5
}
References

https://developers.facebook.com/

https://spark.apache.org/docs/1.0.2/quick-start.html

https://spark.apache.org/documentation.html

http://docs.mongodb.org/manual/

https://docs.python.org/2/tutorial/

http://www.tutorialspoint.com/mongodb/mongodb_quick_guide.htm

https://www.youtube.com/watch?v=o3u97Vry2bE

https://www.youtube.com/watch?v=bWorBGOFBWY&list=PL-x35fyliRwhKT-NpTKprPW1bkbdDcTTW
Research Paper:
Petabyte Scale Databases and Storage Systems Deployed at Facebook
Dhruba Borthakur, Engineer at Facebook.
Four major types of storage systems in Facebook

• Online Transaction Processing Databases (OLTP)
  • The Facebook Social Graph
• Semi-online Light Transaction Processing Databases (SLTP)
  • Facebook Messages and Facebook Time Series
• Immutable DataStore
  • Photos, videos, etc.
• Analytics DataStore
  • Data Warehouse, Logs storage
## Size and Scale of Databases

<table>
<thead>
<tr>
<th></th>
<th>Total Size</th>
<th>Technology</th>
<th>Bottlenecks</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Facebook Graph</strong></td>
<td>Single digit petabytes</td>
<td>MySQL and TAO</td>
<td>Random read IOPS</td>
</tr>
<tr>
<td><strong>Facebook Messages and Time Series Data</strong></td>
<td>Tens of petabytes</td>
<td>HBase and HDFS</td>
<td>Write IOPS and storage capacity</td>
</tr>
<tr>
<td><strong>Facebook Photos</strong></td>
<td>High tens of petabytes</td>
<td>Haystack</td>
<td>storage capacity</td>
</tr>
<tr>
<td><strong>Data Warehouse</strong></td>
<td>Hundreds of petabytes</td>
<td>Hive, HDFS and Hadoop</td>
<td>storage capacity</td>
</tr>
</tbody>
</table>
# Characteristics

<table>
<thead>
<tr>
<th></th>
<th>Query Latency</th>
<th>Consistency</th>
<th>Durability</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Facebook Graph</strong></td>
<td>&lt; few milliseconds</td>
<td>quickly consistent across data centers</td>
<td>No data loss</td>
</tr>
<tr>
<td><strong>Facebook Messages and Time Series Data</strong></td>
<td>&lt; 200 milliseconds</td>
<td>consistent within a data center</td>
<td>No data loss</td>
</tr>
<tr>
<td><strong>Facebook Photos</strong></td>
<td>&lt; 250 milliseconds</td>
<td>immutable</td>
<td>No data loss</td>
</tr>
<tr>
<td><strong>Data Warehouse</strong></td>
<td>&lt; 1 minute</td>
<td>not consistent across data centers</td>
<td>No silent data loss</td>
</tr>
</tbody>
</table>
Facebook Social Graph: TAO and MySQL

• An OLTP workload:
  ▪ Uneven read heavy workload
  ▪ Huge working set with creation-time locality
  ▪ Highly interconnected data
  ▪ Constantly evolving
  ▪ As consistent as possible

• Graph data Model
  • Nodes and Edges: Objects and Associations
Data model

• Objects & Associations
  • Object -> unique 64 bit ID plus a typed dictionary
    ▪ (id) -> (otype, (key -> value)* )
    ▪ ID 6815841748 -> {'type': page, 'name': "John", ... }
  • Association -> typed directed edge between 2 IDs
    ▪ (id1, atype, id2) -> (time, (key -> value)* )
    ▪ (8636146, RSVP, 130855887032173) -> (1327719600, {'response': 'YES'})
  • Association lists
    ▪ (id1, atype) -> all associations with given id1, atype in desc order by time
Architecture: Cache & Storage
Messages & Time Series Database SLTP workload

- Facebook Messages:
Why HBase

• High write throughput
• Horizontal scalability
• Automatic Failover
• Strong consistency within a data center
• Benefits of HDFS: Fault tolerant, scalable, Map-Reduce toolset,

• Why is this SLTP?
  • Semi-online: Queries run even if part of the database is offline
  • Lightweight Transactions: single row transactions
  • Storage capacity bound rather than iops or cpu bound
What we store in HBase

• Small messages
• Message metadata (thread/message indices)
• Search index
• Large attachments stored in Haystack (photo store)
Size and scale of Messages Database

- 6 Billion messages/day
- 74 Billion operations/day
- At peak: 1.5 million operations/sec
- 55% read, 45% write operations
- Average write operation inserts 16 records
- All data is lzo compressed
- Growing at 8 TB/day
Haystack: The Photo Store

- Facebook Photo DataStore

<table>
<thead>
<tr>
<th></th>
<th>2009</th>
<th>2012</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Total Size</strong></td>
<td>15 billion photos 1.5 Petabyte</td>
<td>High tens of petabytes</td>
</tr>
<tr>
<td><strong>Upload Rate</strong></td>
<td>30 million photos/day 3 TB/day</td>
<td>300 million photos/day 30 TB/day</td>
</tr>
<tr>
<td><strong>Serving Rate</strong></td>
<td>555K images/sec</td>
<td></td>
</tr>
</tbody>
</table>
Hive Analytics Warehouse

- Life of a photo tag in Hadoop/Hive storage

**Periodic Analysis (HIVE)**
- Daily report on count of photo tags by country (**1 day**)

**Adhoc Analysis (HIVE)**
- Count photos tagged by females age 20-25 yesterday

**Hive Warehouse**

**Scrapes**
- User info reaches Warehouse (**1 day**)

**MySQL DB**

**Realtime Analytics (HBASE)**
- Count users tagging photos in the last hour (**1 min**)

**Scribe Log Storage (HDFS)**
- Log line reaches Scribeh (**10s**)

**www.facebook.com**
- Log line generated: `<user_id, photo_id>`

**Copier/Loader**
- Log line reaches warehouse (**15 min**)

**Nocron**
- User tags a photo
Why Hive?

• Prospecting for gold in the wild-west.....
  ▪ A platform for huge data-experiments
  ▪ A majority of queries are searching for a single gold nugget
  ▪ Great advantage in keeping all data in one queryable system
  ▪ No structure to data, specify structure at query time

• Crowd Sourcing for data discovery
  ▪ There are 50K tables in a single warehouse
  ▪ Users are DBAs themselves
  ▪ Questions about a table are directed to users of that table
  ▪ Automatic query lineage tools help here
Thank You.