Building a ML System to Predict User Behavior on Mesos

Agenda:

- **Background** on me, Sailthru & Sightlines (mercifully short)
- **Cost effective** resources in the AWS cloud
- **Efficient(ish)** application design
- **Easy** maintenance and evolution
- **Lessons** learned
- **New Innovation**
@jeremystan

- Capitalism
  - Graduate student (Math 2000)
  - Consultant (Finance 2005)
  - CTO (Ad Tech 2010)
  - Chief Data Scientist (Mar Tech 2015)

- Idealism
  - Math 2000

Indirect Value -> Direct Value
Sailthru

Powering More Than 400 Ecommerce & Media Brands

Mashable  The Economist  BIRCHBOX  ALEX AND ANI  FRANK & OAK

POWERED BY

SAILTHRU

1:1 EXPERIENCES  ENGAGEMENT  REVENUE

SAILTHRU
Sightlines

**Analytics**
- Segmentation
- Forecasting

**Personalization**
- Recommendations
- Discounting

**Optimization**
- Frequency
- Channel
Requirements

1. ~5 million users per client
2. JSON formatted user data, siloed across clients
3. Predict varying outcomes
   normal, poisson, binomial, quantile, ...
4. Update models & predictions daily
5. Only really care about predictive performance
6. Scale to 1,000+ clients
Our Cost Effective Scaling Strategy

1. Get really cheap computing power  10x
2. Make it work really, really hard   3x
3. Optimize apps for ease of evolution \[0.6x = \frac{0.2x}{1x}\]
4. Setup identical A/B environments  0.5x

\[0.5x \times 0.6x = 9x\]

Iterate aggressively based on data:
- Features
- Efficiency
- Scale
Cost Effective Resources in the AWS Cloud
Cost Effective

r3.8xlarge
32 vCPU, 244GB RAM

Cost Per Hour

- $2.80 (on demand)
- $1.76 (reserved 1yr)
- $1.05 (reserved 3yr)
- $0.28 (spot instance)
- $9.80 (Cloud)
- $10.50 (Data Center)

Spot + Mesos + Relay
$0.30

30x more cost efficient!

($10.50 = $1.05 / 10%)

Resource Utilization

- 90% (highly efficient)
- 30% (typical cloud)
- 10% (data center)
AWS Spot Instances

Your bid

What you pay

All instances died!
Mesos

Cluster: mesos-dev
Server: 172.24.1.137:5050
Version: 0.22.1
Built: 3 months ago by root
Started: 11 hours ago
Elected: 11 hours ago

LOG

Slaves
Activated 146
Deactivated 0

Tasks
Staged 261,030
Started 0
Finished 178,048
Killed 32,945
Failed 35,029
Lost 2,554

Resources
<table>
<thead>
<tr>
<th></th>
<th>CPUs</th>
<th>Mem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>3,410</td>
<td>25450.2 GB</td>
</tr>
<tr>
<td>Used</td>
<td>2,567.11</td>
<td>23425.9 GB</td>
</tr>
<tr>
<td>Offered</td>
<td>0</td>
<td>0 B</td>
</tr>
<tr>
<td>Idle</td>
<td>842.890</td>
<td>2024.3 GB</td>
</tr>
</tbody>
</table>

146 “agents”
4 availability zones
2 instance types

75% CPU utilized
92% RAM utilized

3,410 CPUs
25TB of RAM

$30 per hour
$260k per year
How we use Mesos

Mesos Agent (16 CPU)

Mesos Agent (8 CPU)

Queue Size

Auto-scales tasks

Relay. Mesos

Framework
How we use Mesos

Mesos Agent (16 CPU)

Mesos Agent (8 CPU)

Queue Size

Relay. Mesos

Stolos

Auto-scales tasks

Dependency Management → Bin Packing

Framework

Tasks
How we use Mesos

Mesos Agent (16 CPU)

Mesos Agent (8 CPU)

Zone 1  Zone 2  Zone 3  Zone 4

- Relay
- Mesos

Queue Size

Relay

Stolos

App Code

Auto-scales tasks

Dependency Management → Bin Packing

Accomplish a small task

Framework

Tasks

Framework

Tasks
How we use Mesos

Zone 1
Zone 2
Zone 3
Zone 4

Mesos Agent (16 CPU)

Mesos Agent (8 CPU)

Relay. Mesos
Stolos
App Code

Queue Size

Auto-scales tasks
Dependency Management → Bin Packing
Accomplish a small task
Mesos + Relay

Before Relay

Available Mesos CPU Jiffies

User

Idle

After Relay

Available Mesos CPU Jiffies

User

Idle

Time

Relay.Mesos
Auto-scaler for distributed applications
[github.com/sailthru/relay.mesos](https://github.com/sailthru/relay.mesos)

- Allocates resources based on queue size
- Wraps applications inside Mesos agents
- Can significantly improve cluster utilization
Efficient(ish) Application Design
Application Pipeline (simplified)
Application Pipeline (actual)

Stolos
Distributed task dependency manager

github.com/sailthru/stolos
- Directed acyclic graph
- Parameterizable templates
- Handles queueing
- Ensures idempotent

Actually much more complex
- ~1,000 clients
- ~10 models
- ~30 steps
- ~100 sub-tasks
Sampling Strategy

JSON sharded on hash(user)
Sampling Strategy

S3

Mongo

JSON

Spark

Day N

shard 1

shard 1,000

Day 1

...
Sampling Strategy

Consistent 0.1% of data to a Mesos Agent CPU

Day N
shard 1

Day 1

shard 1,000

S3

Mongo

JSON

Spark

SAILTHRU
Sampling Strategy

S3

Mongo

JSON

Spark

Apps sample more as needed

Day N

Day 1

shard 1

shard 1,000

...
User Profile JSON Data
Each User Radically Different
Each User Radically Different

**tidyjson**

Turn JSON into data frames

[github.com/sailthru/tidyjson](https://github.com/sailthru/tidyjson)

- Arbitrary JSON into R data.frames
- Guarantees deterministic structure
- Seamless with `dplyr` and `%>%`
What is a Gradient Boosting Machine? (GBM)

1. Build a simple decision tree to predict your response
2. Evaluate it’s performance, and trust it a small amount
3. Build another decision tree to correct it’s mistakes
4. Iterate to some fixed number of trees
Why GBMs?

- **Predict varying outcomes**
  normal, poisson, binomial, quantile, …

- **Flexible enough to capture non-linearity & complex interactions**
  no need to feature engineer for each client

- **Minimal number of hyper-parameters**
  depth, shrinkage, number of trees

- **Robust to missing values**
  no need to impute
Distributing a GBM

\[ \alpha_1 + \alpha_2 + \alpha_3 + \ldots + \alpha_K \]

Diagram:
- Tree 1
- Tree 2
- Tree 3
- Tree K
Distributing a GBM

1. Across the sum
Gives bagging, not boosting (iterative)
=> less accurate
Distributing a GBM

1. Across the sum
   Gives bagging, not boosting (iterative)
   => less accurate

2. Within each tree (Spark MLLib, H20)
   A lot of overhead and coordination
   => not efficient for many small GBMs
Distributing a GBM

1. **Across the sum**
   - Gives bagging, not boosting (iterative)
   - => less accurate

2. **Within each tree** (Spark MLLib, H20)
   - A lot of overhead and coordination
   - => not efficient for many small GBMs

3. **Across the GBMs**
   - 50,000 GBMs to build
   - => each can be built independently

50,000 = 1,000 clients * 10 models * 5-fold CV
For each client & model:

1. Grid search over:
   a. Depth: size of trees
   b. Shrinkage: λ “learning rate” for \( \{ \alpha_i \} \)

2. Cross-validate for optimal # of trees
Easy Maintenance & Evolution
Tools Used

Cluster
- AWS Spot Compute
- Asgard Auto Scaling
- Mesos Sharing

State
- AWS S3 Batch
- Zookeeper Coordination

Maintenance
- ELK Log Mgmt
- Librato Monitoring
- Sensu Alerting

Configuration
- Consul Discovery
- Chef Automation

Frameworks
- Spark Map Reduce
- Marathon Running Apps

Applications
- R Modeling
- Python ETL
How we Iterate

A

Cluster
- AWS Spot
- Aegard
- Mesos

State
- AWS S3
- Basin
- Zookeeper

Maintenance
- ELK
- Log Insight
- Librato Monitoring
- Sensu

Configuration
- Consul
- Chef

Frameworks
- Spark
- MapReduce
- Marathon
- Running Apps

Applications
- R
- Python
- PTL

B

JSON

API

Sailthru User API

Mongo
How we Iterate

A

Cluster
- AWS Spot
- Aagard
- Mesos

State
- AWS S3
- Basin
- Zookeeper

Configuration
- Consul
- Chef

Frameworks
- Spark
- Marathon
- Running Apps

Applications
- R
- Python

Maintenance
- ELK
- Log

B

Cluster
- AWS Spot
- Aagard
- Mesos

State
- AWS S3
- Basin
- Zookeeper

Configuration
- Consul
- Chef

Frameworks
- Spark
- Marathon
- Running Apps

Applications
- R
- Python

Maintenance
- ELK
- Log

Sailthru User API

JSON

API

Mongo
How we Iterate

- Tools
- Configuration
- Applications

API

Sailthru User API

Mongo

v1.0.0
How we Iterate

- Tools
- Configuration
- Applications

v1.0.0

v1.0.1

API

Sailthru User API

Mongo

SAILTHRU
How we Iterate

- Tools
- Configuration
- Applications

v1.0.0

v1.0.1

docker

API

Sailthru User API

Mongo

Cluster
- AWS Spot
- AWS Auto Scaling
- Mesos

State
- AWS S3
- Zookeeper
- Marathon

Maintenance
- ELK
- Log R
- Stack

Framework
- Spark
- Map Reduce

Applications
- Python
- Chef

JSON

A

B
How we Iterate

- Tools
- Configuration
- Applications

- v1.0.0
- v1.0.1
- v1.0.2

API

Sailthru User API
Mongo

JSON

A

B

SAILTHRU
How we Iterate

- Tools
- Configuration
- Applications

A

Sailthru User API

B

Mongo

✓ Check monitoring
How we Iterate

- Tools
- Configuration
- Applications

- v1.0.0
- v1.0.1
- v1.0.2

✓ Check monitoring
✓ Check logging

Sailthru User API
Mongo

API

JSON

A

B

SAILTHRU
How we Iterate

- Tools
- Configuration
- Applications

A

- Check monitoring
- Check logging
- Check performance

B

JSON

Sailthru User API

Mongo
How we Iterate

- Tools
- Configuration
- Applications

- v1.0.0
- v1.0.1
- v1.0.2

API

Sailthru User API

Mongo

✓ Check monitoring
✓ Check logging
✓ Check performance
How we Iterate

- Tools
- Configuration
- Applications

v1.0.0
v1.0.1
v1.0.2

API

Sailthru User API
Mongo

✓ Check monitoring
✓ Check logging
✓ Check performance
Lessons Learned
Lessons Learned

1) **Build multiple layers of fault tolerance**
   - Infrastructure - distributed, redundant
   - Scheduling - 1+ execution and idempotent apps
   - Application - fall back to stale data if need be
Lessons Learned

2) Keep apps and infrastructure isolated and simple
   ● If you can’t explain it in a sentence or need a lot of tests, it’s too complex
     ■ Mesos - resource management
     ■ Zookeeper - consistent cluster state
     ■ Marathon - init process for long-running services
     ■ Relay - task auto-scaling
     ■ Stolos - DAG scheduling
     ■ Consul - infrastructure service discovery
     ■ etc.
Lessons Learned

3) **Bound investments in tools, evolve use or give them up quickly**
   - Marathon - doesn’t handle a huge number of short lived tasks well
   - Chronos - cannot handle thousands of independent DAGs
   - Spark - use only if you really can’t fit your data into RAM
   - HDFS - use S3 if you’re in AWS and design for eventual consistency
Lessons Learned

4) Avoid static partitioning of infrastructure / services / batch
   ● Much more cost effective to pool resources across them all
   ● Design all to be equally tolerant to failures
   ● But must have a means of guaranteeing minimum requirements for some
Lessons Learned

5) **Optimize for innovation**
   - Build a MVP that meets product requirements
   - Focus on redundancy, deployment and monitoring early (get this right)
   - Stay 10x ahead of scale requirements to minimize disruption from “events”
   - Then make iterative infrastructure and app investments to drive ROI
New Innovation
Item Predictions - Reverse Search

Top Recommended

1. Fulton Regular Cotton Pant
   - 22.7% - $98.50

2. Aiden Slim Knit Short
   - 19.5% - $70.00

3. Soft-Wash Bold Plaid Shirt
   - 19.5% - $89.50

4. Tailored Gray Linen Cotton Blazer
   - 17.9% - $230.00

5. Tailored Textured Navy Blazer
   - 16.2% - $230.00
Item Predictions - Methodology

User Profile → User Features → Joint Features → GBM

Item Profile → Item Features → Joint Features → GBM

GBM:

\[ \sum_{k=1}^{K} \alpha_k \]

\[ \Pr( \text{User buys Item} ) \]
Item Predictions - Results
Thank You! Our team:

Divyanshu Vats  Alex Gaudio  Andras Kerekes  Jeremy Stanley  Max Sperlich
Connect with us.

www.sailthru.com
sales@sailthru.com
817.812.8689

NYC HQ
160 Varick St., 12th Floor
New York, NY 10013

San Francisco
25 Taylor St., Room 724
San Francisco, CA 94102

Los Angeles
7083 Hollywood Blvd
Los Angeles, CA 90028

London
18 Soho Square
London, UK, W1D 3QL