Building Data Products with Hadoop

Sam Shah

LinkedIn

February 2, 2011
What is a data product?

A product whose mission is to allow interaction with data in an interesting and meaningful way.
Look before you leap
Tall oaks grow from little acorns
Don’t put the cart before the horse
A stitch in time saves nine
Half a loaf is better than none

Data products at LinkedIn
DATA PRODUCTS AT LINKEDIN

Profile Stats Pro

Who's Viewed My Profile

TODAY

David Feldman
Director, User Experience Design
San Francisco Bay Area | Internet
In Common: 1 shared connection

Kendra Shimmell
Design Research & User Experience Design, Director
Columbus, Ohio Area | Design
In Common: 1 shared connection 3 shared groups

Yevgeniy Urdenko
Experienced Electronic Hardware Designer
San Francisco Bay Area | Consumer Electronics
In Common: 1 shared connection

Someone in the Art & Design function in the Finance Industry from San Francisco Bay Area

Last Week

Software Engineer at Microsoft

Legend

Visits

Appear in Search

Trends

Get tips for a better profile

Top Search Keywords

1. Design 45%
2. UI design 21%
3. User Experience 11%
4. Flash 8%
5. Flex 3%
6. Action Script 3 1%

Viewers by Geography

Top Country Visits
Look before you leap
Tall oaks grow from little acorns
Don’t put the cart before the horse
A stitch in time saves nine
Half a loaf is better than none

**Data products at LinkedIn**

---

**Viewers of this profile also viewed...**

- Russell Jurney
  Data Prodineer at LinkedIn
- Eric Tschetter
  Squirrelly One
- Florian Leibert
  Software Engineer, Research at Twitter
- Lili Wu
  Senior Software Engineer at LinkedIn
- Sam Rash
  Software Engineer at Facebook
- Jay Kreps
  Principal Engineer and Engineering...
- David Phillips
  Software Engineer
- Anmol Bhasin
  Engineering Manager and Senior...
- Baq Haidri
  Senior Software Engineer at LinkedIn
- Pierre-Alexandre Meyer
  Software Engineer - Analytics at Ning

---

**Professionals that recommend this product also recommend...**

- LinkedIn InMaps
- LinkedIn Custom Groups
- LinkedIn Company Pages
- LinkedIn Jobs
- LinkedIn Mobile

---

**People Who Viewed This Job Also Viewed**

- Marketing Manager, Customer Engagement at LinkedIn
- Marketing Manager, Online Channels at LinkedIn
- Relationship Manager, SMB at LinkedIn
- Research Associate at LinkedIn
- Product Marketing Lead, Corporate Recruiting Solutions
DATA PRODUCTS AT LINKEDIN
OVERVIEW

- Take something that runs once...
  - ...and run it multiple times
  - ...and serve it at scale
  - ...and iterate quickly
OVERVIEW

- Take something that runs once...
  - ...and run it multiple times
  - ...and serve it at scale
  - ...and iterate quickly

Using off-the-shelf open source components
Who am I?
Proverbs

Look before you leap

Tall oaks grow from little acorns

Don’t put the cart before the horse

A stitch in time saves nine

Half a loaf is better than none
Look before you leap... 

Do you have a data product?

- Involves ingenuity & creativity
- Research & analysis
Look before you leap...

Do you have a data product?

- Involves ingenuity & creativity
- Research & analysis

Let’s build “People You May Know”
Online vs. Offline

- Sub-second processing
- Harder to scale
- Must handle failures gracefully

- Computationally intensive
- Easier to scale
- Easier to tolerate failures
- Iterate quickly
Online vs. Offline

Online
- Sub-second processing
- Harder to scale
- Must handle failures gracefully

Offline
- Computationally intensive
- Easier to scale
- Easier to tolerate failures
- Iterate quickly
Online vs. Offline

- Sub-second processing
- Harder to scale
- Must handle failures gracefully

- Computationally intensive
- Easier to scale
- Easier to tolerate failures
- Iterate quickly
**PEOPLE YOU MAY KNOW**

How do people know each other?

- **Alice**
- **Bob**
- **Carol**

Known as triangle closing $P \propto \# \text{people known in common}$
PEOPLE YOU MAY KNOW

How do people know each other?

Alice

Bob

Carol
People You May Know

How do people know each other?

Known as triangle closing
How do people know each other?

Known as triangle closing

\[ P(\text{know person}) \propto \# \text{ people known in common} \]
Look before you leap
Tall oaks grow from little acorns
Don’t put the cart before the horse
A stitch in time saves nine
Half a loaf is better than none

**TRIANGLE CLOSING IN Pig**

```pig
/* connections contains (source_id, dest_id) bidirectional pairs */
connections = LOAD 'connections' USING AvroStorage();
by_member = GROUP connections BY source_id;
by_member = FOREACH by_member GENERATE
generatePairs(connections.dest_id) AS (id1, id2);

fof = GROUP by_member BY (id1, id2);
fof = FOREACH fof GENERATE
    flatten(group) AS (source_id, dest_id),
    COUNT(by_member) AS common_neighbors;

STORE fof INTO 'closed-triangles' USING AvroStorage();
```
TRIANGLE CLOSING IN PIG

/* connections contains (source_id, dest_id) bidirectional pairs */
connections = LOAD 'connections' USING AvroStorage();
by_member = GROUP connections BY source_id;
by_member = FOREACH by_member GENERATE
generatePairs(connections.dest_id) AS (id1,id2);

fof = GROUP by_member BY (id1, id2);
fof = FOREACH fof GENERATE
    flatten(group) AS (source_id, dest_id),
    COUNT(by_member) AS common_neighbors;

STORE fof INTO 'closed-triangles' USING AvroStorage();
Triangle closing in Pig

/* connections contains (source_id, dest_id) bidirectional pairs */
connections = LOAD 'connections' USING AvroStorage();
by_member = GROUP connections BY source_id;
by_member = FOREACH by_member GENERATE
generatePairs(connections.dest_id) AS (id1,id2);

fof = GROUP by_member BY (id1, id2);
fof = FOREACH fof GENERATE
    flatten(group) AS (source_id, dest_id),
    COUNT(by_member) AS common_neighbors;

STORE fof INTO 'closed-triangles' USING AvroStorage();
Triangle closing in Pig

/* connections contains (source_id, dest_id) bidirectional pairs */
connections = LOAD 'connections' USING AvroStorage();
by_member = GROUP connections BY source_id;
by_member = FOREACH by_member GENERATE
generatePairs(connections.dest_id) AS (id1,id2);

fof = GROUP by_member BY (id1, id2);
fof = FOREACH fof GENERATE
    flatten(group) AS (source_id, dest_id),
    COUNT(by_member) AS common_neighbors;

STORE fof INTO 'closed-triangles' USING AvroStorage();
TRIANGLE CLOSING IN Pig

/* connections contains (source_id, dest_id) bidirectional pairs */
connections = LOAD 'connections' USING AvroStorage();
by_member = GROUP connections BY source_id;
by_member = FOREACH by_member GENERATE
generatePairs(connections.dest_id) AS (id1,id2);

fof = GROUP by_member BY (id1, id2);
fof = FOREACH fof GENERATE
flatten(group) AS (source_id, dest_id),
COUNT(by_member) AS common_neighbors;

STORE fof INTO 'closed-triangles' USING AvroStorage();
Look before you leap  Tall oaks grow from little acorns  Don’t put the cart before the horse  A stitch in time saves nine  Half a loaf is better than none

Triangle closing in Pig

/* connections contains (source_id, dest_id) bidirectional pairs */
connections = LOAD 'connections' USING AvroStorage();
by_member = GROUP connections BY source_id;
by_member = FOREACH by_member GENERATE
  generatePairs(connections.dest_id) AS (id1,id2);

fof = GROUP by_member BY (id1, id2);
fof = FOREACH fof GENERATE
  flatten(group) AS (source_id, dest_id),
  COUNT(by_member) AS common_neighbors;

STORE fof INTO 'closed-triangles' USING AvroStorage();
TRIANGLE CLOSING EXAMPLE

1. \{A \rightarrow B, B \rightarrow A, B \rightarrow C, C \rightarrow B\}
TRIANGLE CLOSING EXAMPLE

1. \{A \rightarrow B, B \rightarrow A, B \rightarrow C, C \rightarrow B\}
2. A : \{B\}, B : \{A, C\}, C : \{B\}
**TRIANGLE CLOSING EXAMPLE**

1. \{A \rightarrow B, B \rightarrow A, B \rightarrow C, C \rightarrow B\}
2. \[ A : \{B\}, B : \{A, C\}, C : \{B\} \]
Tall oaks grow from little acorns
Don’t put the cart before the horse
A stitch in time saves nine
Half a loaf is better than none

TRIANGLE CLOSING EXAMPLE

1. \{A \to B, B \to A, B \to C, C \to B\}
2. A : \{B\}, B : \{A, C\}, C : \{B\}
3. B : \{A \to C, C \to A\}
TRIANGLE CLOSING EXAMPLE

1. \{A \rightarrow B, B \rightarrow A, B \rightarrow C, C \rightarrow B\}
2. A : \{B\}, B : \{A, C\}, C : \{B\}
3. B : \{A \rightarrow C, C \rightarrow A\}
4. \{(A \rightarrow C, 1); (C \rightarrow A, 1)\}
Our Workflow
Our Workflow

triangle-closing

top-n
Our Workflow

- triangle-closing
- top-n
- push-production
Look before you leap

Tall oaks grow from little acorns

Don’t put the cart before the horse

A stitch in time saves nine

Half a loaf is better than none
Our Workflow

triangle-closing → top-n → push-production
Look before you leap  
Tall oaks grow from little acorns  
Don’t put the cart before the horse  
A stitch in time saves nine  
Half a loaf is better than none

**Our Workflow**

1. triangle-closing
2. remove-connected
3. top-n
4. push-production
Our Workflow

- Triangle-closing
- Remove-connected
- Top-n
- Push-production
- Push-qa
PYMK WORKFLOW
WHAT WE NEED . . .

- Dependency management
- Diverse job types
- Scheduling
- Monitoring
- Configuration
- Retry/restart on failure
- Resource locking
- Log collection
- Historical information
What we need...

- Dependency management
- Diverse job types
- Scheduling
- Monitoring
- Configuration
- Retry/restart on failure
- Resource locking
- Log collection
- Historical information
Example Azkaban Job Spec

type=pig
pig.script=top-n.pig

param.N=50

dependencies=triangle-closing
Look before you leap
Tall oaks grow from little acorns
Don’t put the cart before the horse
A stitch in time saves nine
Half a loaf is better than none

Flow Instance

Name: push-production Flow ID: 2304

Start Time: 02-01-2011 11:29:04
End Time: 02-01-2011 11:29:10
Period: 0 minutes
Look before you leap
tall oaks grow from little acorns
don't put the cart before the horse
A stitch in time saves nine
half a loaf is better than none
Look before you leap
Tall oaks grow from little acorns
Don’t put the cart before the horse
A stitch in time saves nine
Half a loaf is better than none
Look before you leap

Tall oaks grow from little acorns

Don’t put the cart before the horse

A stitch in time saves nine

Half a loaf is better than none

PROVERBS
STORAGE SYSTEM CONSIDERATIONS

- Cost of data load?
- Data stored per node? Response time?
- Fail-over?
- How to transfer?
- Versioning & rollback?
Look before you leap
Tall oaks grow from little acorns
Don’t put the cart before the horse
A stitch in time saves nine
Half a loaf is better than none

STORAGE SYSTEM CONSIDERATIONS

- Cost of data load?
- Data stored per node? Response time?
- Fail-over?
- How to transfer?
- Versioning & rollback?
Voldemort RO extensions

- Partition data over cluster of machines
- Build store offline in Hadoop
- Fast query times
- Replication with automatic failover
- Versions new datasets & allows rollback
Look before you leap

Tall oaks grow from little acorns

Don’t put the cart before the horse

A stitch in time saves nine

Half a loaf is better than none
Veriﬁcation

Data issues notoriously hard to solve

- Check your invariants
- Check your invariants
  - Before & after processing
  - Statistical checks helpful too
Look before you leap
Tall oaks grow from little acorns
Don’t put the cart before the horse
A stitch in time saves nine
Half a loaf is better than none

VERIFICATION

Data issues notoriously hard to solve

► Check your invariants
► Check your invariants
  ► Before & after processing
  ► Statistical checks helpful too

What if connections data contained duplicates?
Data Quality

- Create dataset with explanatory info
  - Push to QA cluster
  - Viewer applications—EXPLAIN command
- Versioning
  - Have ability to rollback quickly
- Unit tests!
PROFILE YOUR JOBS

static class TheReducer extends /* ... */ {
    @Override
    public void configure(JobConf job) {
        myData = processFromDistributedCache();
        /* ... */
    }
    /* ... */
}

conf.setCombinerClass(TheReducer.class);

  ▶ Ran in production for a while

Ran in production for a while

m · r reads from distributed cache
m = 5000, r = 2000 ⇒ ten million calls!

≈ 50% overhead of job
PROFILE YOUR JOBS

static class TheReducer extends /* ... */ {
    @Override
    public void configure(JobConf job) {
        myData = processFromDistributedCache();
        /* ... */
    }
    /* ... */
}

conf.setCombinerClass(TheReducer.class);

- Ran in production for a while
- \( m \cdot r \) reads from distributed cache
  - \( m = 5000, r = 2000 \implies \) ten million calls!
  - \( \approx 50\% \) overhead of job
Proverbs

Look before you leap

Tall oaks grow from little acorns

Don’t put the cart before the horse

A stitch in time saves nine

Half a loaf is better than none
IMPROVING PERFORMANCE

- Symmetry
  - Alice knows Carol, Carol knows Alice
IMPROVING PERFORMANCE

- **Symmetry** 50% speedup
  - Alice knows Carol, Carol knows Alice
TRIANGLE CLOSING IN Pig

/* connections contains (source_id, dest_id) bidirectional pairs */
connections = LOAD 'connections' USING AvroStorage();
by_member = GROUP connections BY source_id;
by_member = FOREACH by_member GENERATE
generatePairs(connections.dest_id) AS (id1,id2);

fof = GROUP by_member BY (id1, id2);
fof = FOREACH fof GENERATE
   flatten(group) AS (source_id, dest_id),
   COUNT(by_member) AS common_neighbors;

STORE fof INTO 'closed-triangles' USING AvroStorage();
IMPROVING PERFORMANCE

- **Symmetry**  50% speedup
  - Alice knows Carol, Carol knows Alice

- **Limiting**
  - Ignore members with $> k$ connections
IMPROVING PERFORMANCE

- **Symmetry**  50% speedup
  - Alice knows Carol, Carol knows Alice
- **Limiting**
  - Ignore members with $> k$ connections
- **Sampling**
  - Randomly choose up to $k$ connections
IMPROVING PERFORMANCE

▶ **Symmetry**  Structural (incl. Incremental)
  ▶ Alice knows Carol, Carol knows Alice

▶ **Limiting**  Bounding
  ▶ Ignore members with $> k$ connections

▶ **Sampling**  Probabilistic
  ▶ Randomly choose up to $k$ connections
IMPROVING PERFORMANCE

- **Symmetry**  Structural (incl. Incremental)
  - Alice knows Carol, Carol knows Alice
- **Limiting**  Bounding
  - Ignore members with $> k$ connections
- **Sampling**  Probabilistic
  - Randomly choose up to $k$ connections
PROBABILISTIC FILTERING

- Biggest bottleneck is intermediate I/O

Build Bloom filter of existing connections
- No false negatives
- Some false positives
- Filter map-side
**PROBABILISTIC FILTERING**

- Biggest bottleneck is intermediate I/O
PROBABILISTIC FILTERING

- Biggest bottleneck is intermediate I/O

- Build Bloom filter of existing connections
  - No false negatives
  - Some false positives

- Filter map-side

Map Phase

Reduce Phase
PROBABILISTIC FILTERING

triangle-closing

remove-connected

top-n

push-production

push-qa

▶ Build Bloom filter of existing connections
▶ No false negatives
▶ Some false positives
▶ Filter map-side
PROBABILISTIC FILTERING

- Build Bloom filter of existing connections
  - No false negatives
  - Some false positives
- Filter map-side
TODAY...

- Take something that runs once...
  - ...and run it multiple times
  - ...and serve it at scale
  - ...and iterate quickly
SNA TEAM

http://sna-projects.com
http://sna-projects.com/blog

Abhishek Gupta
Adam Smyczek
Adil Aijaz
Albert Wang
Alex Feinberg
Alexis Pribula
Ali Imam
Anmol Bhasin
Baoqiu Cui
Baq Haidri
Cheng-Tao Chu
Chris Conrad
Chris Riccomini
Christian Posse
DJ Patil
Daniel Tukelang
David McCutcheon
Eric Tschetter
Ethan Zhang
Fatih Emekci
Hao Yan
Heyning Cheng
Igor Perisic
Jakob Homan
Janet Ryu
Jay Kreps
Jeffrey Schang
Jill Chen
Jingwei Wu
Jiong Wang
John Wang
Jonathan Koren
Joseph Adler
Joshua Hartman
Jun Rao
Lili Wu
Lin Guo
Mathieu Bastian
Matthew Hayes
Micah Alpern
Monica Rogati
Neha Narkhede
Parul Jain
Paul Ogilvie
Peter Skomoroch
Ramesh Dommeti
Richard Park
Ron Bekkerman
Roshan Sumbaly
Rui Wang
Russell Jurney
Sam Shah
Shannon Zhang
Utku Irmak
Wei Shen
Yasuhiro Matsuda