Scaling up data science applications

How switching to Spark improved performance, realizability and reduced cost

Kexin Xie
Director, Data Science

kexin.xie@salesforce.com
@realstraw

Yacov Salomon
VP, Data Science

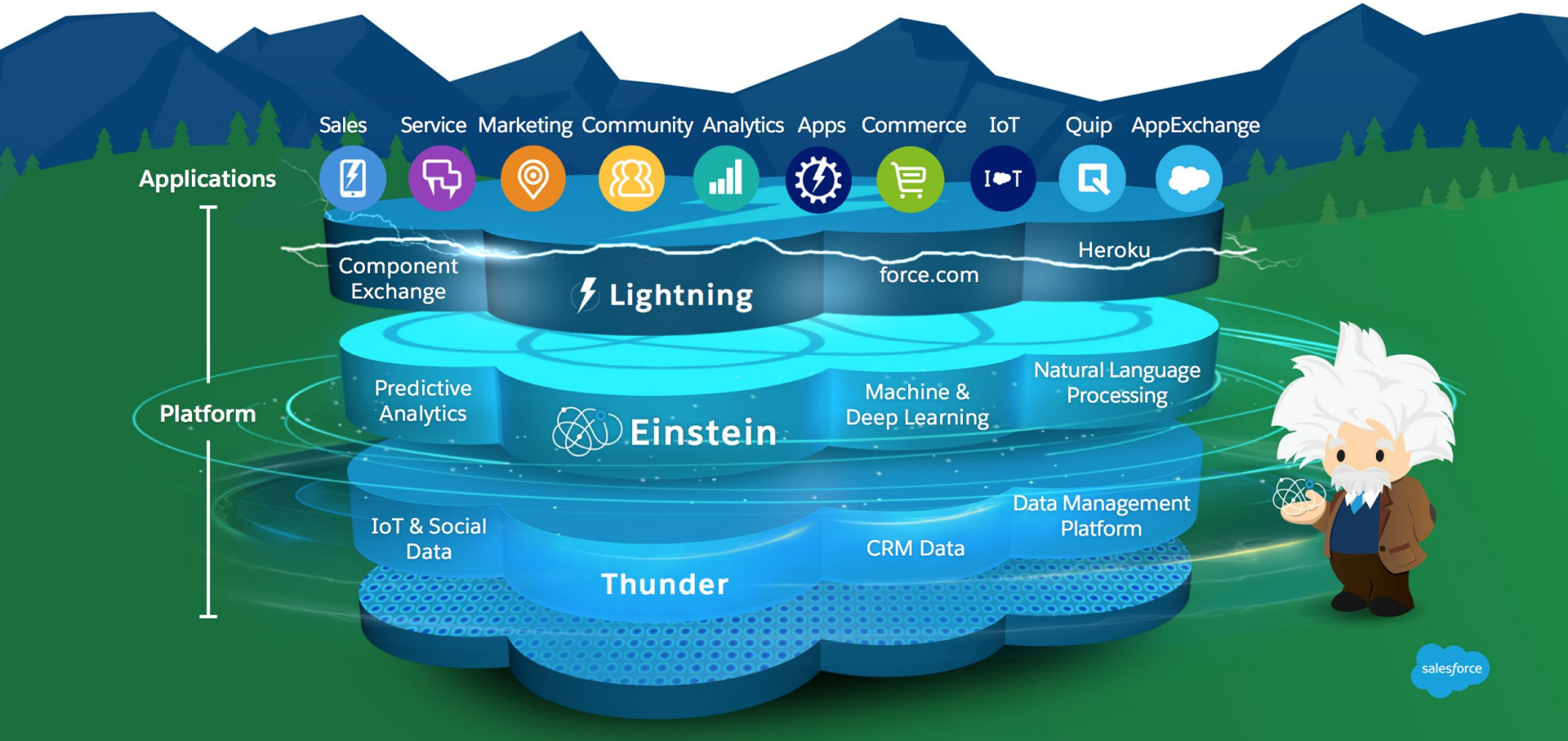
ysalomon@salesforce.com

Fastest Growing Top 5 Enterprise Software Company

"Innovator of the Decade"

Forbes September 2016

Intelligent Customer Success Platform




salesforce
salesforce DMP
Right Now


Boom
Coca-Cola
L'ORÉAL
Red Bull
LOWE'S
Nestlé
Mondelēz International
CONAGRA BRANDS
Kellogg's
AB InBev
Heineken
Reckitt Benckiser
PEUGEOT
KEURIG
HERSHEY'S
Hotels.com
GP Georgia-Pacific
LIONSGATE
turner
DURACELL
ticketmaster
pandora
NBCUniversal
Campbell's
Hallmark
SONY
E*TRADE
Carrefour
CARMAX
Expedia
Liberty Mutual
jetBlue
KITH ACE
UNDER ARMOUR
The New York Times
Time Inc.
YAHOO!
MAJOR LEAGUE BASEBALL
cars.com
HBO
2015-2016
Casper
BUSINESS INSIDER
meredith
News Corp
Slate
match
Bloomberg
VICE
BBC
Spotify
SOCIETE GENERALE
priceline.com

In an Internet Minute

4.1 million
videos viewed

342,000
apps downloaded

156 million
emails sent

452,000
tweets sent

40,000
hours listened

3.5 million
search queries

900,000
logins

$751,522
spent online

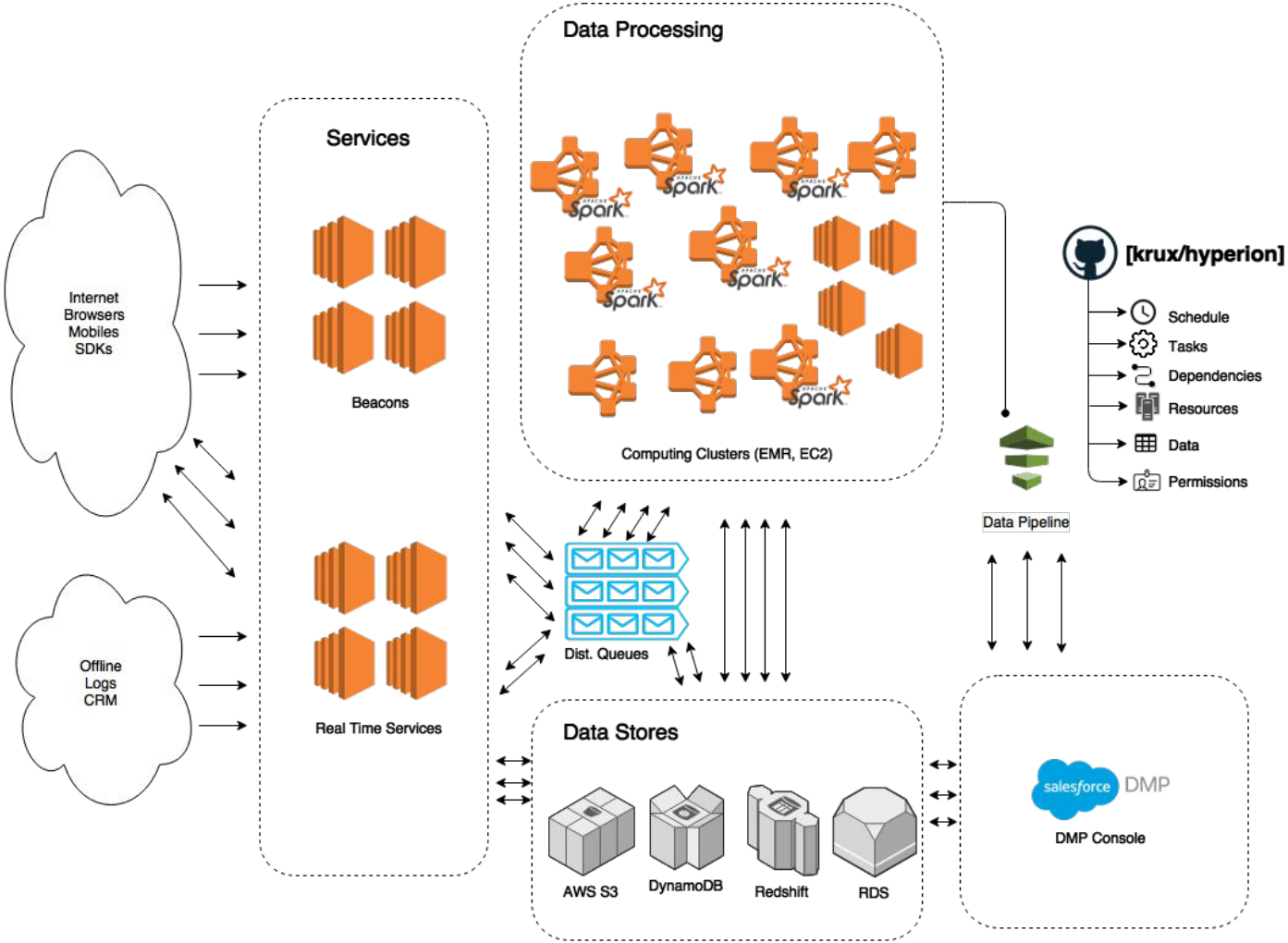
Salesforce DMP is at Internet Scale

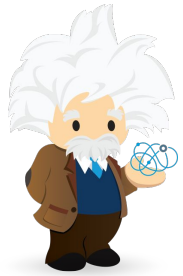
4.2 million
user match requests

1.6 million
page views

4.75 million
data capture events

700,000
Ad impressions


Data Processing
Services
Internet
Browsers
Mobiles
SDKs
Offline
Logs
CRM
Beacons
Real Time Services
Computing Clusters (EMR, EC2)
[krux/hyperion]
Schedule
Tasks
Dependencies
Resources
Data
Permissions
Data Pipeline
Dist. Queues
Data Stores
AWS S3
DynamoDB
Redshift
RDS
salesforce DMP
DMP Console



Marketers want to find more customers like their loyal customer

LOYAL CUSTOMER

- AGE: MALE 22-30
- INCOME: $250K+
- LUXURY AUTO SPENDER
- TECH-SAVVY
- SPORTS ENTHUSIAST
- BUSINESS EXEC
- UNIVERSITY/POST-GRADUATE

REACH

Lookalikes

salesforce

Email Newsletter Subscriber — Standard Segment / ID: rtwch9s8p
18.5M — 4.9M — Last Modified — Type — Category — Sub-Category: Attributes

Vail Season Ticket holder — Standard Segment / ID: rtwch9uio
14.4M — 3. — Sub-Category: Primary

JetBlue SLC, Denver, Jackson Hole Fliers — Standard Segment / ID: rtwch9v92
5.7M — 1.5 — Sub-Category: Primary

Lonely Planet Adventure Traveler — Standard Segment / ID: rtwch9xi6
14.4M — 3. — Sub-Category: Primary

Hiking Customers - TAM — Standard Segment / ID: rtwch9yz6
93.2M — 38 — Sub-Category: Primary

Hiking Customers - High Propensity — Standard Segment / ID: rtwch90hs
28.3M — 11 — Sub-Category: Primary

Email Engaged - Hiking Customers — Standard Segment / ID: rtwch91po
Last Modified: May 16, 2017 — Type: Clusters — Category: Behavioral — Sub-Category: Attributes

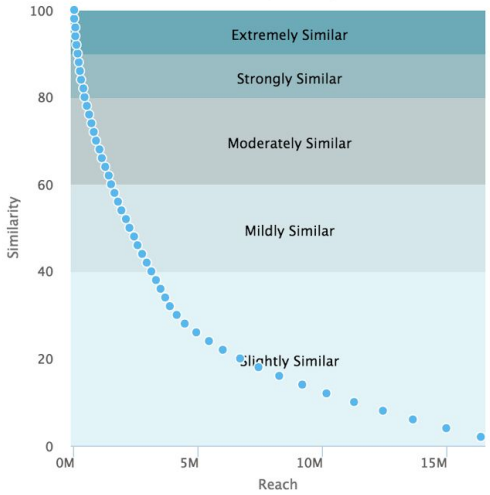
Email Non-Engaged - Hiking Customers — Standard Segment / ID: rtwch921r
3.8M — 987.1K — Last Modified: May 18, 2017 — Type: Clusters — Category: Behavioral — Sub-Category: Attributes

Create lookalikes for Lonely Planet Adventure Traveler

Select a point to create lookalike segments.

Lookalike segments

you can zoom into the chart by selecting a chart area



Reach	Similarity
5,603	100%
18,968	98%
41,317	96%
73,335	94%
112,160	92%
152,564	90%
193,435	88%
237,750	86%
291,049	84%
355,732	82%

Close

Help


Model
Naive Bayes Framework

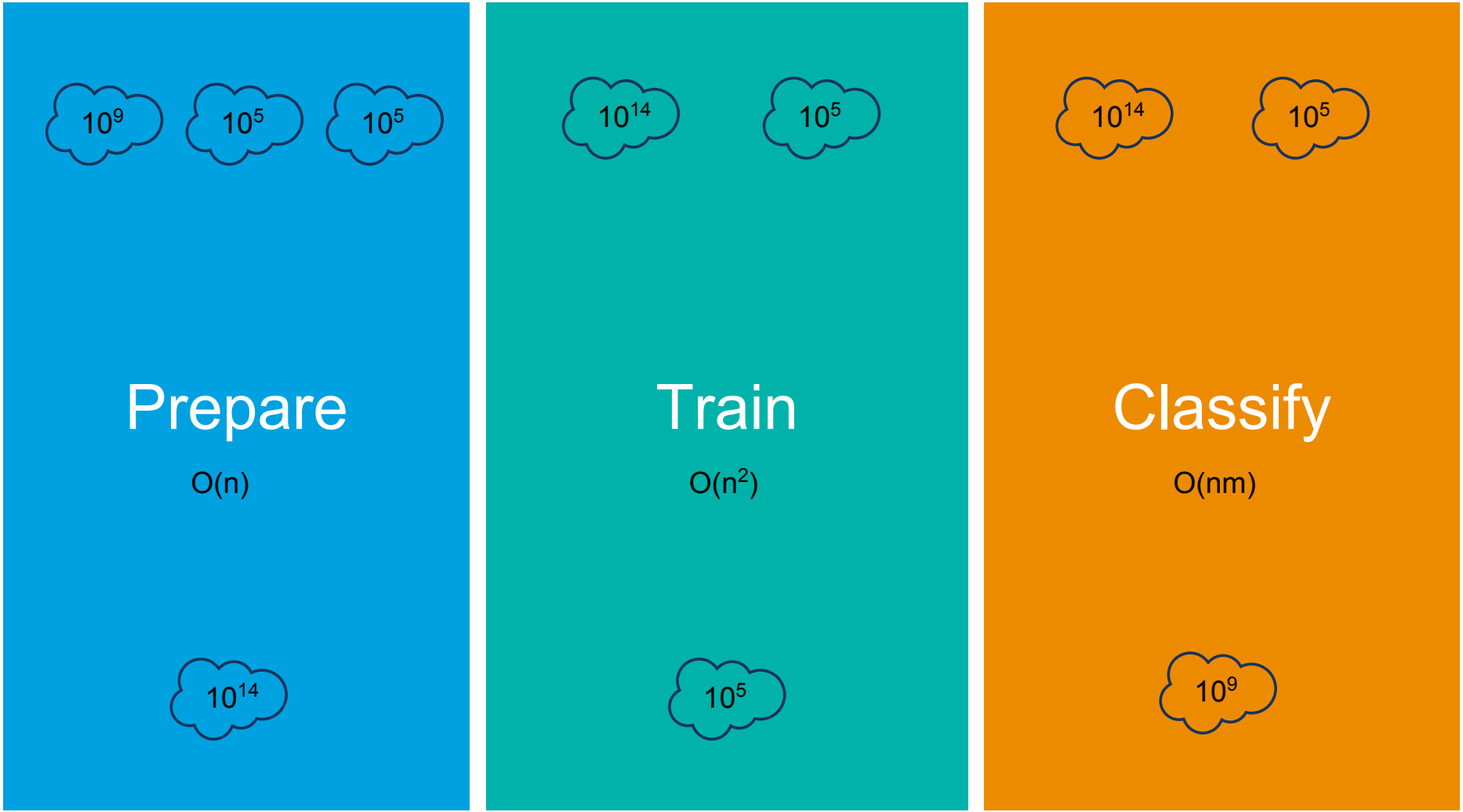
Model	Naive Bayes Framework
Feature Selection	Linear Discriminant Analysis

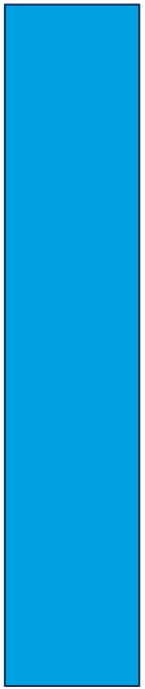
Model	Naive Bayes Framework
Feature Selection	Linear Discriminant Analysis
Science / Art	Correct for autocorrelation in feature space (paper pending)

Prepare

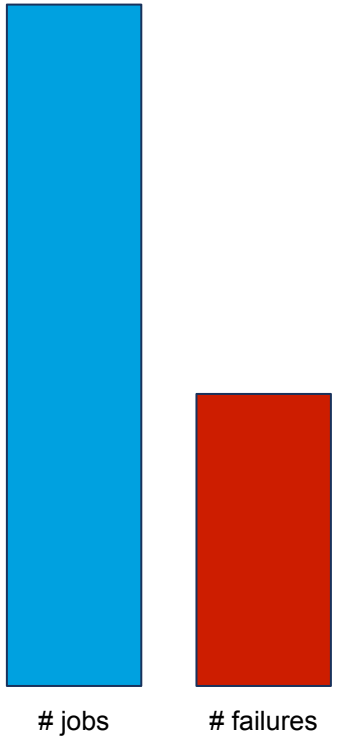
Train

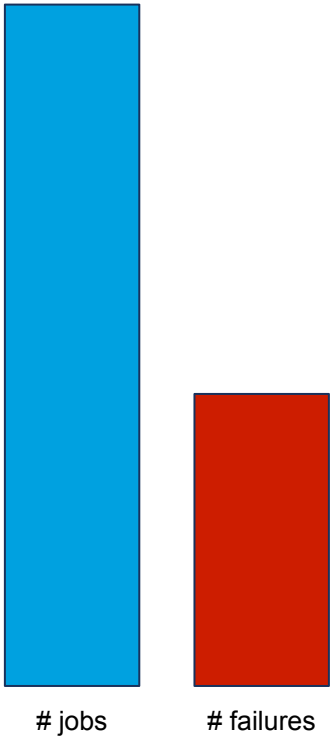
Classify


10^9
10^5
10^5
Prepare
O(n)
10^{14}
10^{14}
10^5
Train
$O(n^2)$
10^5
10^{14}
10^5
Classify
O(nm)
10^9



jobs


jobs
failures



StackOverflowException

Job Stuck

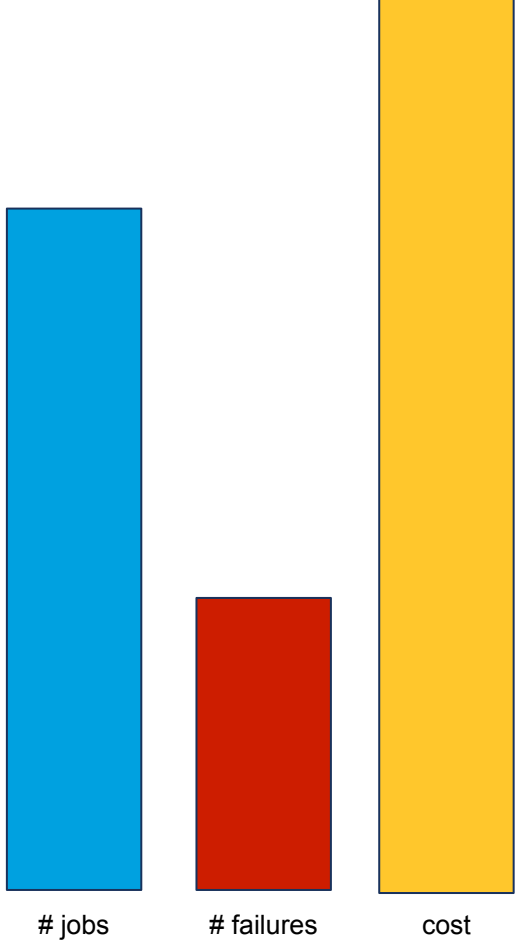
Slave Node Keeps Dying

Idle Slave Nodes

Code Complexity

Out Of Memory

Intermediate Result Serialization



StackOverflowException

Job Stuck

Slave Node Keeps Dying

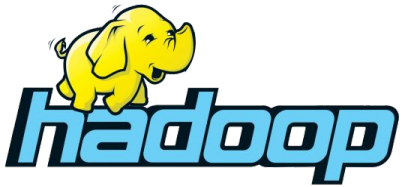
Idle Slave Nodes

Code Complexity

Out Of Memory

Intermediate Result Serialization

Reality: Framework



Big I/O Cost

Code Complexity

Flexibility

Number of Features

Total Population

Segment Populations

Segment Population Overlap

```java
public static class Map1 extends Mapper<LongWritable, Text, Text, NullWritable> {

    private Text segment = new Text();

    public void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException {
        UserSegments userSegments = UserSegments.parse(value.toString());
        segment.set(userSegments.getUserId);
        context.write(userSegments, NullWritable.get());
    }
}

public static class Reduce1 extends Reducer<Text, NullWritable, Text, NullWritable> {

    public void reduce(Text key, Iterable<NullWritable> values, Context context)
            throws IOException, InterruptedException {
        context.write(key, NullWritable.get());
    }
}

public static class Map2 extends Mapper<LongWritable, Text, Text, NullWritable> {

    private final static IntWritable one = new IntWritable(1);

    public void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException {
        context.write(NullWritable.get(), one);
    }

}

public static class Reduce2 extends Reducer<NullWritable, IntWritable, IntWritable, NullWritable> {

    public void reduce(NullWritable key, Iterable<IntWritable> values, Context context)
            throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }
        context.write(new IntWritable(sum), NullWritable.get());
    }
}
```

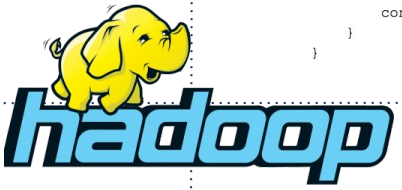
```java
public static class Map
        extends Mapper<LongWritable, Text, NullWritable, LongWritable> {

    private final static LongWritable one = new LongWritable(1);

    public void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException {
        context.write(NullWritable.get(), one);
    }
}

public static class Reduce
        extends Reducer<NullWritable, LongWritable, LongWritable, NullWritable> {

    public void reduce(NullWritable key, Iterable<LongWritable> values, Context context)
            throws IOException, InterruptedException {
        long sum = 0;
        for (LongWritable val : values) {
            sum += val.get();
        }
        context.write(new LongWritable(sum), NullWritable.get());
    }
}
```




```java
public static class Map
        extends Mapper<LongWritable, Text, Text, LongWritable> {
    private final static LongWritable one = new LongWritable(1);
    private Text segment = new Text();

    public void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException {
        String line = value.toString();
        UserSegments userSegments = UserSegments.parse(value.toString());
        for (String seg : userSegments.segments) {
            segment.set(seg);
            context.write(seg, one);
        }
    }

}

public static class Reduce
        extends Reducer<Text, LongWritable, Text, LongWritable> {

    public void reduce(Text key, Iterable<LongWritable> values, Context context)
            throws IOException, InterruptedException {
        int sum = 0;
        for (LongWritable val : values) {
            sum += val.get();
        }
        context.write(key, new LongWritable(sum));
    }

}
```

```java
public static class Map
        extends Mapper<LongWritable, Text, Text, LongWritable> {
    private final static LongWritable one = new LongWritable(1);
    private Text segmentPair = new Text();

    public void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException {
        String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);
        UserSegments userSegments = UserSegments.parse(value.toString());
        for (String seg1 : userSegments.segments) {
            for (String seg2 : userSegments.segments) {
                segmentPair.set(seg1 + "," + seg2);
                context.write(segmentPair, one);
            }
        }
    }
}

public static class Reduce extends
        Reducer<Text, LongWritable, Text, LongWritable> {

    public void reduce(Text key, Iterable<LongWritable> values, Context context)
            throws IOException, InterruptedException {
        int sum = 0;
        for (LongWritable val : values) {
            sum += val.get();
        }
        context.write(key, new LongWritable(sum));
    }
}
```

```
userSegments
  .flatMap(_.segments)
  .distinct
  .count
```

```
userSegments.count
```


```
userSegments
  .flatMap(r => r.segments.map(_ -> 1L))
  .reduceByKey(_ + _)
```

```
val userSegmentPairs = userSegments
  .flatMap(r => r.segments.map(r.userId -> _))

userSegmentPairs
  .join(userSegmentPairs)
  .map { case (_, (feat1, feat2)) => (feat1, feat2) -> 1L }
  .reduceByKey(_ + _)
```

Reality: Data in many S3 prefixes/folders

```
val inputData = Seq(
  "s3://my-bucket/some-path/prefix1/" ,
  "s3://my-bucket/some-path/prefix2/" ,
  "s3://my-bucket/some-path/prefix2/" ,
  ...
  "s3://my-bucket/some-path/prefix2000/" ,
)
```

How about this?

```
val myRdd = inputData
  .map(sc.textFile)
  .reduceLeft(_ ++ _)
```

Or this?

```
val myRdd = sc.union(inputData.map(sc.textFile))
```

Solution

```
// get the s3 objects
val s3Objects = new AmazonS3Client()
  .listObjects("my-bucket", "some-path")
  .getObjectSummaries()
  .map(_.getKey())
  .filter(hasPrefix1to2000)
// send them to slave nodes and retrieve content
val myRdd = sc
  .parallelize(Random.shuffle(s3Objects.toSeq), parallelismFactor)
  .flatMap( key =>
    Source
      .fromInputStream(
        new AmazonS3Client().getObjectForKey("my-bucket", key)
          .getObjectContent
      )
      .getLines
  )
```

Reality: Large Scale Overlap

```
val userSegmentPairs = userSegments
  .flatMap(r => r.segments.map(r.userId -> _))

userSegmentPairs
  .join(userSegmentPairs)
  .map { case (_, (feat1, feat2)) => (feat1, feat2) ->  1L }
  .reduceByKey(_ + _)
```

user1	a, b, c
user2	a, b, c
user3	a, b, c
user4	a, c
user5	a, c

user1	a
user1	b
user1	c
user2	a
user2	b
user2	c
user3	a
user3	b
user3	c
user4	a
user4	c
user5	a
user5	c

user1	a	b
user1	a	c
user1	b	c
user2	a	b
user2	a	c
user2	b	c
user3	a	b
user3	a	c
user3	b	c
user4	a	c
user5	a	c

1	a	b
1	a	c
1	b	c
1	a	b
1	a	c
1	b	c
1	a	b
1	a	c
1	b	c
1	a	c
1	a	c

a	b	3
a	c	5
b	c	3

user1	a, b, c
user2	a, b, c
user3	a, b, c
user4	a, c
user5	a, c

hash1	a, b, c	3
hash2	a, c	2

hash1	a	3
hash1	b	3
hash1	c	3
hash2	a	2
hash2	c	2

hash1	a	b	3
hash1	a	c	3
hash1	b	c	3
hash2	a	c	2

a	b	3
a	c	5
b	c	3

Solution

```
// Reduce the user space
val aggrUserSegmentPairs = userSegmentPairs
  .map(r => r.segments -> 1L)
  .reduceByKey(_ + _)
  .flatMap { case (segments, count) =>
    segments.map(s => (hash(segments), (segment, count))
  }

aggrUserSegmentPairs
  .join(aggrUserSegmentPairs)
  .map { case (_, (seg1, count), (seg2, _)) =>
    (seg1, seg2) -> count
  }
  .reduceByKey(_ + _)
```

Reality: Perform Join on Skewed Data

user1	a
user2	b
user3	c
user4	d
user5	e

X

user1	one
user1	two
user1	three
user1	four
user1	five
user1	six
user3	seven
user3	eight
user4	nine
user5	ten

```
data1.join(data2)
```

Executor 1

user1	a

user1	one
user1	two
user1	three
user1	four
user1	five
user1	six

Executor 2

user3	c
user4	d

user3	seven
user3	eight
user4	nine

Executor 3

user5	e
user2	b

user5	ten

Executor 1

user1	salt1	a

user1	salt1	one
user1	salt1	two

Executor 2

user1	salt2	a

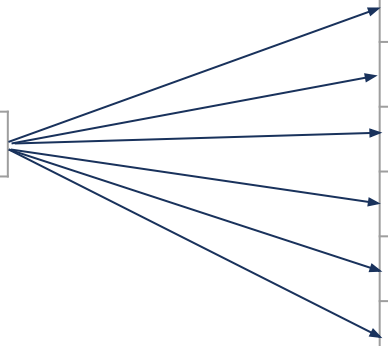
user1	salt2	three
user1	salt2	four

Executor 3

user1	salt3	a

user1	salt3	five
user1	salt3	six

user1	a



user1	one
user1	two
user1	three
user1	four
user1	five
user1	six

user2	b
user3	c
user4	d
user5	e

X

user3	seven
user3	eight
user4	nine
user5	ten

Solution

```
val topKeys = data2
  .mapValues(x => 1L)
  .reduceByKey(_ + _)
  .takeOrdered(10)(Ordering[(String, Long)].on(_._2).reverse)
  .toMap
  .keys

val topData1 = sc.broadcast(
  data1.filter(r => topKeys.contains(r._1)).collect.toMap
)

val bottomData1 = data1.filter(r => !topKeys.contains(r._1))

val topJoin = data2.flatMap {  case (k, v2) =>
  topData1.value.get(k).map(v1 => k -> (v1, v2))
}

topJoin ++ bottomData1.join(data2)
```

Hadoop to Spark → Maintainable codebase

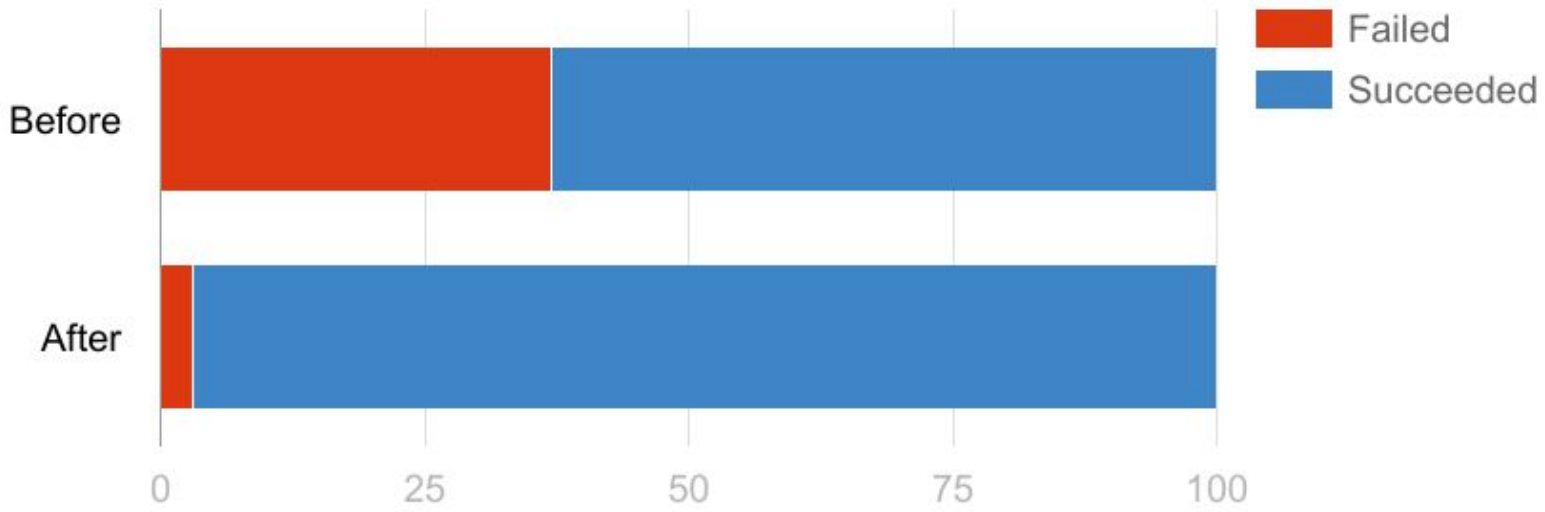
Smarter retrieval of data from S3 → Clients with more than 2000 S3 prefixes/folders

Before: 5 hours
After: 20 minutes

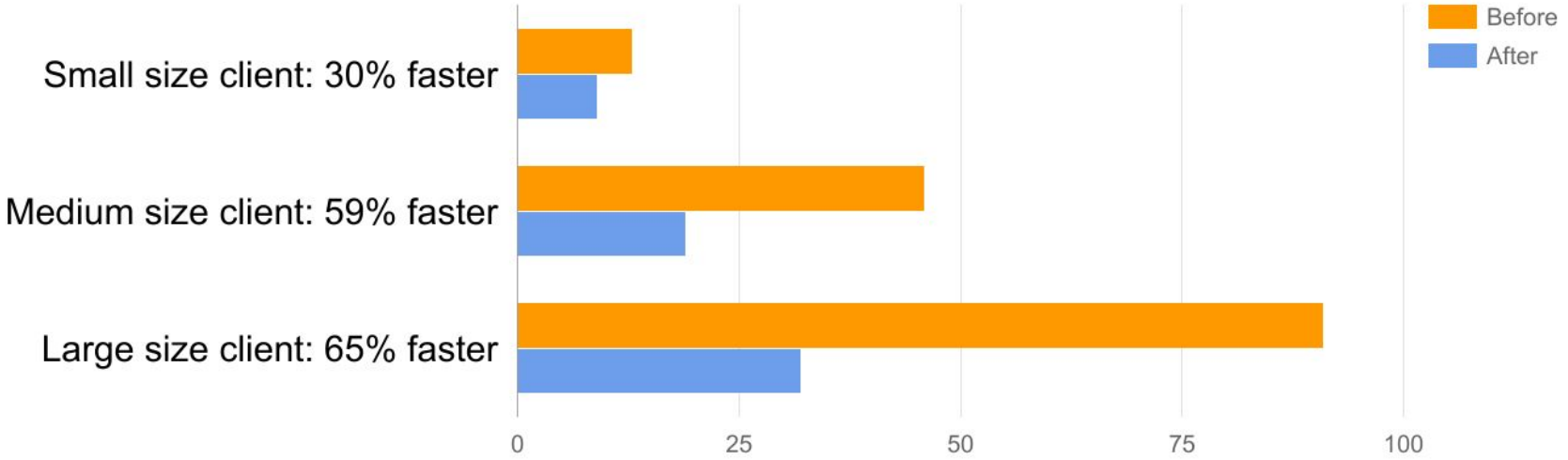
Condensed overlap algorithm → 100x faster and 10x less data for segment overlap

Hybrid join algorithm → Able to process joins for highly skewed data

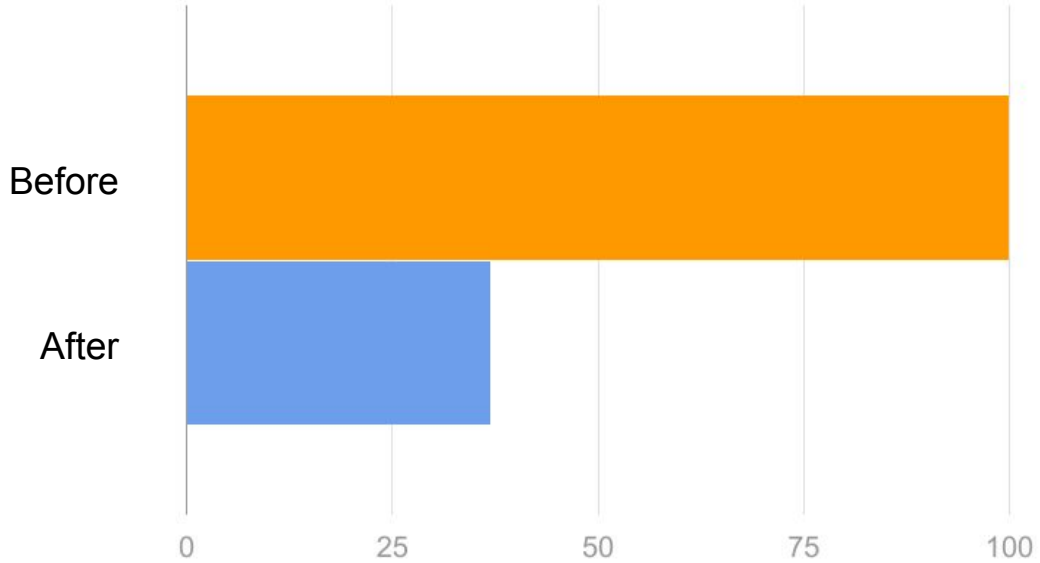
Failure Rate



Performance



Cost



Thank You