INTRODUCTION TO SCALABLE TOOLS FOR BIG DATA
WHO IS THIS GUY?
4 LETTERS, 3 VOWELS?!
What do I do?
About the Boa Language and Infrastructure

Boa is a domain-specific language and infrastructure that eases mining software repositories. Boa's infrastructure leverages distributed computing techniques to execute queries against hundreds of thousands of software projects very efficiently.

Getting Started With Boa in 5 Minutes

An Example Mining Task

Consider answering a question such as "what are the average number of changed files per revision (churn rate) for all projects?"

Answering this question ordinarily requires knowledge of (at a minimum): mining project metadata, mining code repository locations, how to access those code repositories, additional filtering code, controller logic, etc.

```
1 # what are the churn rates for all projects
2 p: Project = input;
3 counts: output mean[string] of int;
```
ENOUGH ABOUT ME...

WHY ARE WE HERE TODAY?
What Happens in an Internet Minute?

- 639,800 GB of global IP data transferred
- 20 New victims of identity theft
- 204 million Emails sent
- 1,300 New mobile users
- 47,000 App downloads
- 100+ New LinkedIn accounts
- $83,000 In sales
- 20 million Photo views
- 61,141 Hours of music
- 320+ New Twitter accounts
- 3,000 Photo uploads
- 61,141 New tweets
- 277,000 Logins
- 6 million Facebook views
- 2+ million Search queries
- 1.3 million Video views
- 20 hours of video uploaded

And Future Growth is Staggering

Today, the number of networked devices = the global population
By 2015, the number of networked devices = 2x the global population
In 2015, it would take you 5 years to view all video crossing IP networks each second
2019: This Is What Happens In An Internet Minute

- 1 Million Logging In
- 18.1 Million Texts Sent
- 4.5 Million Videos Viewed
- 390,030 Apps Downloaded
- 347,222 Scrolling Instagram
- 87,500 People Tweeting
- 1 Million Swipes
- 188 Million Emails Sent
- 4.8 Million Gifs Served
- 2.1 Million Snaps Created
- 41.6 Million Messages Sent
- $996,956 Spent Online
- 694,444 Hours Watched
- 3.8 Million Search Queries
- 180 Smart Speakers Shipped
- 41 Music Streaming Subscriptions

Created By:
@LoriLewis
@OfficiallyChadd
## Big Data & AI Landscape 2018

### Infrastructure
- Hadoop on Premise
- Hadoop in the Cloud
- Streaming / In-Memory

### Analytics
- Data Analyst Platforms
- Data Science Platforms

### Applications - Enterprise
- Sales
- Marketing
- Customer Service

### Applications - Industry
- Healthcare
- Life Sciences
- Transportation

### Cross-Infrastructure/Analytics

### Open Source
- Framework
- Query / Data Floor
- Data Access
- Coordination
- Streaming
- AI / Machine Learning / Deep Learning
- Search

### Data Sources & APIs
- Data Sources & APIs
- Health
- IOT
- Financial & Economic Data
- Air / Space / Sea
- People / Entities
- Location Intelligence
- Other

### Data Resources
- Data Services & Incubators
- Research

---

© Matt Turck (@matturck), Demiade Obayomi (@demi_obayomi), & FirstMark (@firstmarkcap) - matturck.com/bigdata2018
FRAMEWORK

Hadoop
HDFS
MapReduce
YARN
Flink
TEZ
MESOS
Spark
Apache Kylin
SLIDE

QUERY /

Spark
Spark SQL
presto
SLAMData
Google Cloud
HADOOP
DISTRIBUTED
FILE
SYSTEM
BACKGROUND AND TERMINOLOGY
WHAT FEATURES DOES HDFS HAVE?
FAULT TOLERANCE
SCALABILITY

2 Nodes

24 Cores 256GB 3TB
(2 nodes min config)

16 Nodes

576 Cores 24TB 194TB
(16 nodes max config)
DESIGNED FOR PROCESSING DATA
DESIGNED FOR PROCESSING DATA
DESIGNED FOR PROCESSING DATA
DESIGNED FOR PROCESSING DATA
DESIGNED FOR PROCESSING DATA
DESIGNED FOR PROCESSING DATA
DESIGNED FOR PROCESSING DATA
HOW DOES HDFS WORK?

HDFS ARCHITECTURE
MASTER-SLAVE ARCHITECTURE

Master: **Namenode**
- maintains **metadata** about files

Slave(s): **Datanode**
- runs on **each node in cluster**
- stores data in **local file systems**
files split into blocks
(default 128MB)
where can I write?

Client

NN

DN1
DN2
DN3
DN4
Data is replicated (3 times)
Data is replicated (3 times)
Fault Recovery
Fault Recovery

DN1  

DN2  

DN3  

DN4  

NN
Fault Recovery

DN1

DN2

DN3

DN4
HOW TO USE HDFS

- 3 ways
  - Command line interface (‘hdfs dfs' command)
  - Programmatically (Python, Java, Scala, etc)
  - Web interfaces (Hue, etc)
EXAMPLE COMMANDS

• hdfs dfs -ls /input

• hdfs dfs -put ipsum.txt /input/

• hdfs dfs -cat /input/ipsum.txt

• hdfs dfs -rm -r /input/

Note: all commands begin ‘hdfs dfs’
COMMON PITFALLS WITH HDFS
LOTS OF SMALL FILES
LOTS OF SMALL FILES

SequenceFile File Layout

Data

| Key | Value | Key | Value | Key | Value |

MapFile File Layout

Index

| Key | Key |

Data

| Key | Value | Key | Value | Key | Value | Key | Value | Key | Value | Key | Value |
TRYING TO WRITE TO FILES

HDFS files are read only*

(*more or less)
MISCONFIGURATION

- Each DataNode should have multiple physical drives
- HDFS should be using each drive
- Using hardware redundancy (RAID)
HISTORY

• Google published in 2004
• Inspired by LISP map() and reduce() functions
• Apache Hadoop (2006, Yahoo)
• Hadoop 2.x, aka YARN (~2012)
map function
map function

Input:  (key, value)
Output: list of (ikey, ivalue)
map function

Input: (key, value)
Output: list of (ikey, ivalue)

reduce function
map function

Input: (key, value)
Output: list of (ikey, ivalue)

reduce function

Input: (ikey, list of ivalue)
Output: list of (okey, ovalue)
Step 1: Split
(Framework)

Step 2: Map

Step 3: Collect & Sort
(Framework)
FAULT TOLERANCE

• if a task fails (or is really slow)
  • spawns same task on another node

• retries tasks 3 times
HADOOP ACTIVITY
HADOOP ACTIVITY


• Look at your numbers

• Write down how often each number appears
HADOOP ACTIVITY


- Look at your numbers
- Write down how often each number appears
- Give list of numbers and how often they appear to person on the end of the row
HADOOP ACTIVITY

Person at the end of the row:

• Generate final list of numbers and how often they appeared
EXAMPLE: COUNTING WORDS

Input  lines of text

Output  list of words and how often they appeared
MAP FUNCTION

```python
for line in input:
    words = [s.strip() for s in re.split('[\s]', line) if s]
    for word in words:
        print (word, 1)
```
REDUCE FUNCTION

for word, counts in input:
    print (word, sum(counts))
COMMON PITFALLS WITH MAPREDUCE
TRYING TO USE MAP REDUCE FOR EVERYTHING
NOT USING COMBINERS
COMBINERS
## COMBINERS

<table>
<thead>
<tr>
<th>word, 1</th>
<th>word, 1</th>
<th>word, 1</th>
<th>word, 1</th>
<th>word, 1</th>
<th>word, 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
</tr>
<tr>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
</tr>
<tr>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
</tr>
<tr>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
</tr>
<tr>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
</tr>
<tr>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
</tr>
<tr>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
</tr>
<tr>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
</tr>
<tr>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
</tr>
<tr>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
</tr>
<tr>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
</tr>
<tr>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
</tr>
<tr>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
</tr>
<tr>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
</tr>
<tr>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
</tr>
<tr>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
</tr>
<tr>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
</tr>
<tr>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
</tr>
<tr>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
<td>word, 1</td>
</tr>
<tr>
<td>word, 10</td>
<td>word, 10</td>
<td>word, 10</td>
<td>word, 10</td>
<td>word, 10</td>
<td>word, 10</td>
</tr>
</tbody>
</table>
NOT CHANGING
NUMBER OF REDUCERS
APACHE SPARK
HISTORY

- Research project in 2009
- Open sourced and made Apache project
- Most actively developed open source project in Hadoop ecosystem
Data Sharing in MapReduce

Input → HDFS read → iter. 1 → HDFS write → iter. 2 → HDFS write → ...

Input → query 1 → HDFS read → result 1

Input → query 2 → HDFS read → result 2

Input → query 3 → HDFS read → result 3

...
Data Sharing in MapReduce

Slow due to replication, serialization, and disk IO
Data Sharing in Spark

Input

iter. 1

iter. 2

... iter.

Distributed memory

Input

one-time processing

query 1

query 2

query 3

... queries

Input
Data Sharing in Spark

Input

iter. 1

iter. 2

iter. 3

one-time processing

Distributed memory

query 1

query 2

query 3

10-100× faster than network and disk
KEY IDEA

RESILIENT DISTRIBUTED DATASET (RDD)
RDD

- Distributed collections
- Can be cached in memory (spill to disk if too large)
- Manipulated to generate new RDDs
- RDDs form a graph of operations
- Automatically rebuilt on failure (fault tolerant)
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns.
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = sc.textFile('hdfs://...')
errors = lines.filter(lambda x: x.startsWith('ERROR'))
messages = errors.map(lambda x: x.split('\t')[2])
cachedMsgs = messages.cache()
```
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```python
lines = sc.textFile('hdfs://...')
errors = lines.filter(lambda x: x.startswith('ERROR'))
messages = errors.map(lambda x: x.split('	')[2])
cachedMsgs = messages.cache()
```
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```python
lines = sc.textFile('hdfs://...')
errors = lines.filter(lambda x: x.startsWith('ERROR'))
messages = errors.map(lambda x: x.split('	')[2])
cachedMsgs = messages.cache()
```
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```python
lines = sc.textFile('hdfs://...')
errors = lines.filter(lambda x: x.startsWith('ERROR'))
messages = errors.map(lambda x: x.split('	')[2])
cachedMsgs = messages.cache()
```
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```python
lines = sc.textFile('hdfs://...')
errors = lines.filter(lambda x: x.startsWith('ERROR'))
messages = errors.map(lambda x: x.split('	')[2])
cachedMsgs = messages.cache()
```
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

\[
\begin{align*}
\text{lines} &= \text{sc.textFile('hdfs://...')} \\
\text{errors} &= \text{lines.filter(lambda x: x.startsWith('ERROR'))} \\
\text{messages} &= \text{errors.map(lambda x: x.split('t')[2])} \\
\text{cachedMsgs} &= \text{messages.cache()} \\
\text{cachedMsgs.filter(lambda x: x.contains('foo')).count()} 
\end{align*}
\]
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```python
lines = sc.textFile('hdfs://...')
errors = lines.filter(lambda x: x.startswith('ERROR'))
messages = errors.map(lambda x: x.split('	')[2])
cachedMsgs = messages.cache()

cachedMsgs.filter(lambda x: x.contains('foo')).count()
```
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```python
lines = sc.textFile('hdfs://...')
errors = lines.filter(lambda x: x.startswith('ERROR'))
messages = errors.map(lambda x: x.split('	')[2])
cachedMsgs = messages.cache()

cachedMsgs.filter(lambda x: x.contains('foo')).count()
```
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```python
lines = sc.textFile('hdfs://...')
errors = lines.filter(lambda x: x.startsWith('ERROR'))
messages = errors.map(lambda x: x.split('	')[2])
cachedMsgs = messages.cache()

cachedMsgs.filter(lambda x: x.contains('foo')).count()
```
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```python
lines = sc.textFile('hdfs://...')
errors = lines.filter(lambda x: x.startswith('ERROR'))
messages = errors.map(lambda x: x.split('t')[2])
cachedMsgs = messages.cache()

cachedMsgs.filter(lambda x: x.contains('foo')).count()
```
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```python
lines = sc.textFile('hdfs://...')
errors = lines.filter(lambda x: x.startswith('ERROR'))
messages = errors.map(lambda x: x.split('	')[2])
cachedMsgs = messages.cache()

cachedMsgs.filter(lambda x: x.contains('foo')).count()
```
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```python
lines = sc.textFile('hdfs://...')
errors = lines.filter(lambda x: x.startsWith('ERROR'))
messages = errors.map(lambda x: x.split('	')[2])
cachedMsgs = messages.cache()

cachedMsgs.filter(lambda x: x.contains('foo')).count()
```
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```python
lines = sc.textFile('hdfs://...')
errors = lines.filter(lambda x: x.startswith('ERROR'))
messages = errors.map(lambda x: x.split('	')[2])
cachedMsgs = messages.cache()

cachedMsgs.filter(lambda x: x.contains('foo')).count()
```
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```python
lines = sc.textFile('hdfs://...')
errors = lines.filter(lambda x: x.startsWith('ERROR'))
messages = errors.map(lambda x: x.split('\t')[2])
cachedMsgs = messages.cache()

cachedMsgs.filter(lambda x: x.contains('foo')).count()
```
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```python
lines = sc.textFile('hdfs://...')
errors = lines.filter(lambda x: x.startsWith('ERROR'))
messages = errors.map(lambda x: x.split('	')[2])
cachedMsgs = messages.cache()

cachedMsgs.filter(lambda x: x.contains('foo')).count()
cachedMsgs.filter(lambda x: x.contains('bar')).count()
```
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```python
lines = sc.textFile('hdfs://...')
errors = lines.filter(lambda x: x.startsWith('ERROR'))
messages = errors.map(lambda x: x.split('\t')[2])
cachedMsgs = messages.cache()

cachedMsgs.filter(lambda x: x.contains('foo')).count()
cachedMsgs.filter(lambda x: x.contains('bar')).count()
```
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns.

```
lines = sc.textFile('hdfs://...')
errors = lines.filter(lambda x: x.startsWith('ERROR'))
messages = errors.map(lambda x: x.split('	')[2])
cachedMsgs = messages.cache()

cachedMsgs.filter(lambda x: x.contains('foo')).count()
cachedMsgs.filter(lambda x: x.contains('bar')).count()
...
```
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```python
lines = sc.textFile('hdfs://...')
errors = lines.filter(lambda x: x.startswith('ERROR'))
messages = errors.map(lambda x: x.split('\t')[2])
cachedMsgs = messages.cache()

cachedMsgs.filter(lambda x: x.contains('foo')).count()
cachedMsgs.filter(lambda x: x.contains('bar')).count()
...
```

**Result:** full-text search of Wikipedia in <1 sec (vs 20 sec for on-disk data)
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```python
lines = sc.textFile('hdfs://...')
errors = lines.filter(lambda x: x.startsWith('ERROR'))
messages = errors.map(lambda x: x.split('	')[2])
cachedMsgs = messages.cache()

cachedMsgs.filter(lambda x: x.contains('foo')).count()
cachedMsgs.filter(lambda x: x.contains('bar')).count()
...
```

Result: scales to 1 TB data in 5-7 sec (vs 170 sec for on-disk data)
## Supported Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Operator</th>
<th>Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>map</td>
<td>reduce</td>
<td>sample</td>
</tr>
<tr>
<td>filter</td>
<td>count</td>
<td>cogroup</td>
</tr>
<tr>
<td>groupBy</td>
<td>reduceByKey</td>
<td>take</td>
</tr>
<tr>
<td>sort</td>
<td>groupByKey</td>
<td>partitionBy</td>
</tr>
<tr>
<td>join</td>
<td>first</td>
<td>pipe</td>
</tr>
<tr>
<td>leftOuterJoin</td>
<td>union</td>
<td>save</td>
</tr>
<tr>
<td>rightOuterJoin</td>
<td>cross</td>
<td>...</td>
</tr>
</tbody>
</table>
SPARK ACTIVITY
GROUP 1

PASS ON ONLY PAPER WITH EVEN NUMBER TO GROUP 2
GROUP 2

COUNT HOW MANY NUMBERS AND GIVE COUNT TO GROUP 4

PASS ALL NUMBERS TO GROUP 3
ADD ALL THE NUMBERS
GIVE SUM TO GROUP 4
GROUP 4

COMPUTE THE AVERAGE
(SUM / COUNT)
WHAT DID WE JUST DO?

• We were a Spark program!

• Group 1: \[ \text{evens} = \text{data}.\text{filter}(\lambda x: x \ mod \ 2 == 0) \]

• Group 2: \[ \text{evens\_count} = \text{evens}.\text{count}() \]

• Group 3: \[ \text{sum} = \text{evens}.\text{reduce}(\lambda a, b: a + b).\text{collect}() \]

• Group 4: \[ \text{average} = \text{sum} / \text{evens\_count} \]
EXAMPLE: COUNTING WORDS

Input: lines of text

Output: list of words and how often they appeared
READ THE INPUT FILE

data = sc.textFile('sample.txt')
words = data.flatMap(
    lambda line: [s.strip() for s in re.split('\s', line) if s]
)
COUNT THE WORDS

counts = words.map(lambda x: (x, 1))

totals = counts.reduceByKey(lambda a, b: a + b)
SHOW THE RESULT

totals.collect()

or

totals.saveAsTextFile('out.txt')
COMMON PITFALLS WITH SPARK
MAP() VS FLATMAP()

• Important if your map generates a sequence/array

• map() will return an RDD of arrays
  • e.g., [[1, 2], [2, 3], [4], [5, 6, 7]]

• flatMap() returns an RDD of values
  • e.g., [1, 2, 2, 3, 4, 5, 6, 7]
RUNNING ON YARN
NOT CONTROLLING # OF EXECUTORS
NOT CONTROLLING EXECUTOR MEMORY
CONCLUSION