(A brief tour of)

The Magic Behind Spark

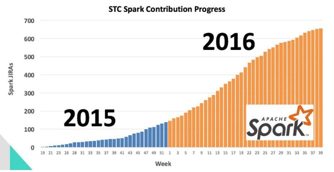
Holden Karau
@holdenkarau

Who am I?

- My name is Holden Karau
- Prefered pronouns are she/her
- I’m a Principal Software Engineer at IBM’s Spark Technology Center
- Apache Spark committer
- previously Alpine, Databricks, Google, Foursquare & Amazon
- co-author of High Performance Spark & Learning Spark (+ more)
- Twitter: @holdenkarau
- Slideshare http://www.slideshare.net/hkarau
- Linkedin https://www.linkedin.com/in/holdenkarau
- Github https://github.com/holdenk
- Related Spark Videos http://bit.ly/holdenSparkVideos

Spark.tc

Spark Technology Center



IBM Spark Technology Center

Founded in 2015.

Location:

Physical: 505 Howard St., San Francisco CA

Web: http://spark.tc   Twitter: @apachespark_tc

Mission:

Contribute intellectual and technical capital to the Apache Spark community.

Make the core technology enterprise- and cloud-ready.

Build data science skills to drive intelligence into business applications — http://bigdatauniversity.com

Key statistics:

About 50 developers, co-located with 25 IBM designers.

Major contributions to Apache Spark http://jiras.spark.tc

Apache SystemML is now an Apache Incubator project.

Founding member of UC Berkeley AMPLab and RISE Lab

Member of R Consortium and Scala Center

Who do I think you all are?

- Nice people*
- Possibly some knowledge of Apache Spark?
- Interested in understanding a bit about how Spark works?
- Want to make your spark jobs more efficient
- Familiar-ish with Scala or Java or Python

Why people come to Spark:

Well this MapReduce job is going to take 16 hours - how long could it take to learn Spark?

Why people come to Spark:

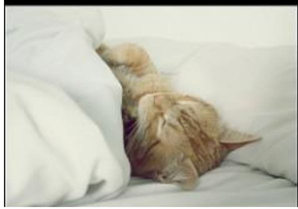
My DataFrame won’t fit in memory on my cluster anymore, let alone my MacBook Pro :(Maybe this Spark business will solve that...

Plus a little magic :)

What is the “magic” of Spark?

- DAG / “query plan” is the root of much of it
  - Think the person behind the curtain
- RDDs: Optimizer to pipeline steps
- Resiliency: recover from failures rather than protecting from failures.
- “In-memory” + “spill-to-disk”
- Functional programming to build the DAG for “free”
- Select operations without deserialization

Not enough time for all the magic



- Some of it was covered yesterday in the morning’s Spark talk
- If you missed that talk the rest of the magic is also in my GOTO Chicago talk - https://gotochgo.com/2017/sessions/33 (slides & video)

Your data is magically distributed

- At some point the RDD or DataFrame is forced to exist
- Then Spark splits up the data on a bunch of different machines
- The default looks “like”* your input data source (often)
- If the data needs to be joined (or similar) Spark does a “shuffle” so it knows which keys are where
- Partioners in Spark are deterministic on key input (e.g. for any given key they must always send to the same partition)

When we say distributed we mean...

Key-skew to the anti-rescue… :(

- Keys aren’t evenly distributed
  - Sales by zip code, or records by city, etc.
- groupByKey will explode (but it's pretty easy to break)
- We can have really unbalanced partitions
  - If we have enough key skew sortByKey could even fail
  - Stragglers (uneven sharding can make some tasks take much longer)
  - We can add some noise if we need to

(94110, A, B)	(94110, A, R)	(94110, E, R)
(94110, A, C)	(10003, A, R)	(67843, T, R)
(10003, D, E)	(94110, D, R)	(94110, T, R)
(94110, E, F)	(94110, E, R)	(94110, T, R)

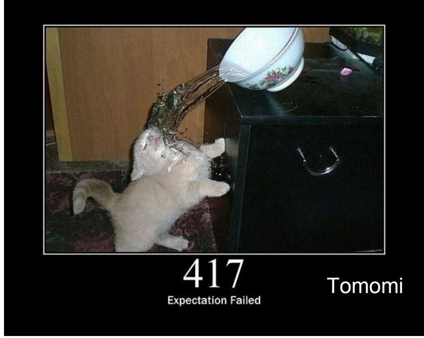
RDDs + lambdas = Black boxes

- Spark can’t see inside your lambdas
- Spark can optimize the maps / flatmaps/ reduceByeKey / etc - but not the things inside of that.
- If you load data in then only access some fields or filter Spark can’t use that information :(

key-skew + black boxes == more sadness

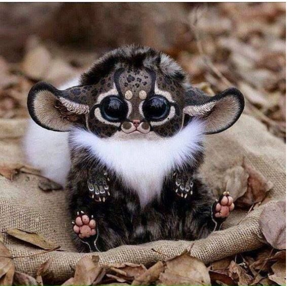
- There is a worse way to do WordCount
- We can use the seemingly safe thing called groupByKey
- Then compute the sum
- But since it’s on a slide of “more sadness” we know where this is going...

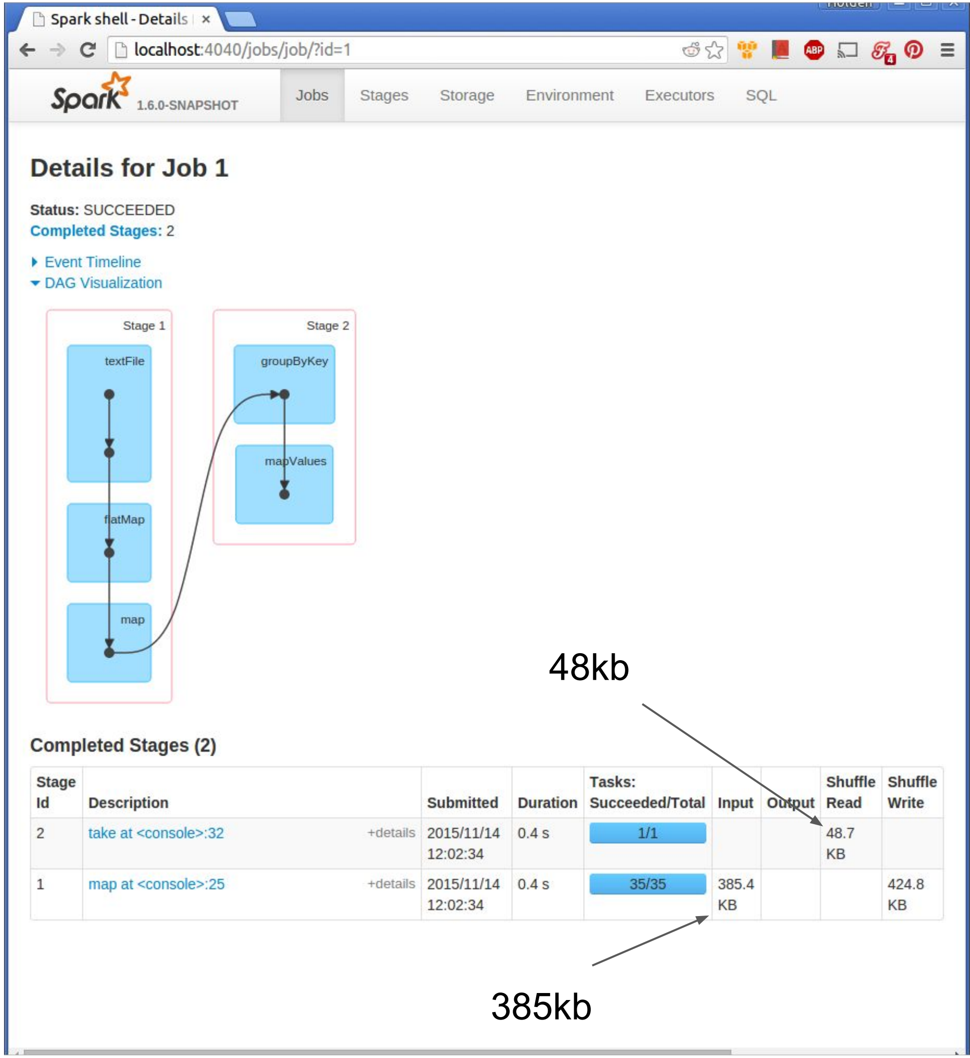
Bad word count :(




```
words = rdd.flatMap(lambda x: x.split(" "))
wordPairs = words.map(lambda w: (w, 1))
grouped = wordPairs.groupByKey()
counted_words = grouped.mapValues(lambda counts: sum(counts))
counted_words.saveAsTextFile("boop")
```

GroupByKey

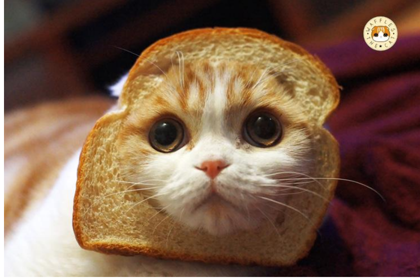




48kb

385kb

So what did we do instead?



- reduceByKey
  - Works when the types are the same (e.g. in our summing version)
- aggregateByKey
  - Doesn’t require the types to be the same (e.g. computing stats model or similar)

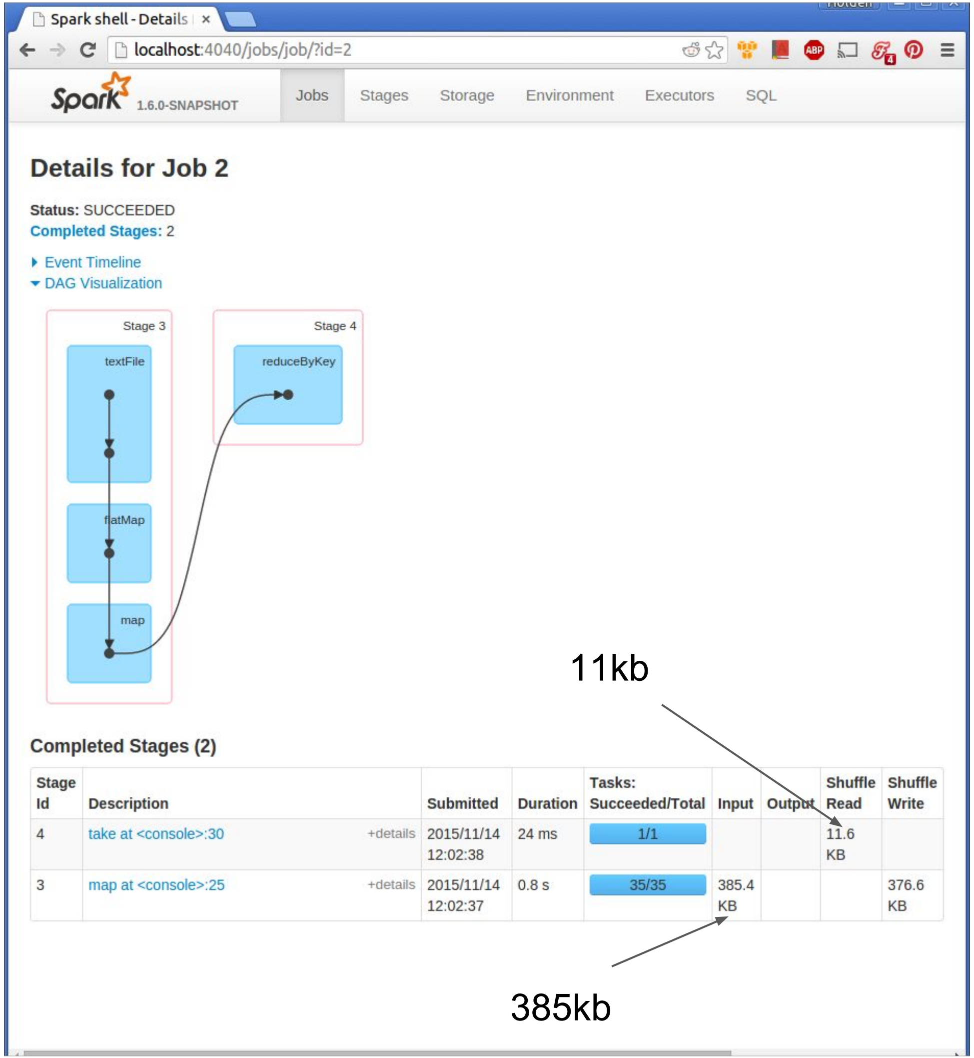
Allows Spark to pipeline the reduction & skip making the list

We also got a map-side reduction (note the difference in shuffled read)

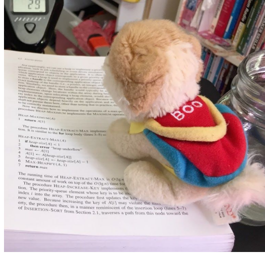
Effectively allows Spark to “understand” our operation more

Note: we can’t use the “noise” approach from the shuffle tricks to replace groupByKey

reduceByKey



Opening the black box: Datasets / DFs



- Operations *can* be written in a **DSL Spark can understand**
- Can “escape” back to RDDs and arbitrary lambdas
- But give you more options to “help” the optimized
- groupBy returns a GroupedDataStructure and offers special aggregates
- Selects can push filters down for us*
- Magic codegen for certain statements :D
- Etc.

Using Datasets to mix functional & relational


```
ds.filter($"happy" === true).
  select($"attributes"(0).as[Double]).
  reduce((x, y) => x + y)
```

Traditional functional reduction:
arbitrary scala code :)

A typed query (specifies the return type). Without the as[] will return a DataFrame (Dataset[Row])

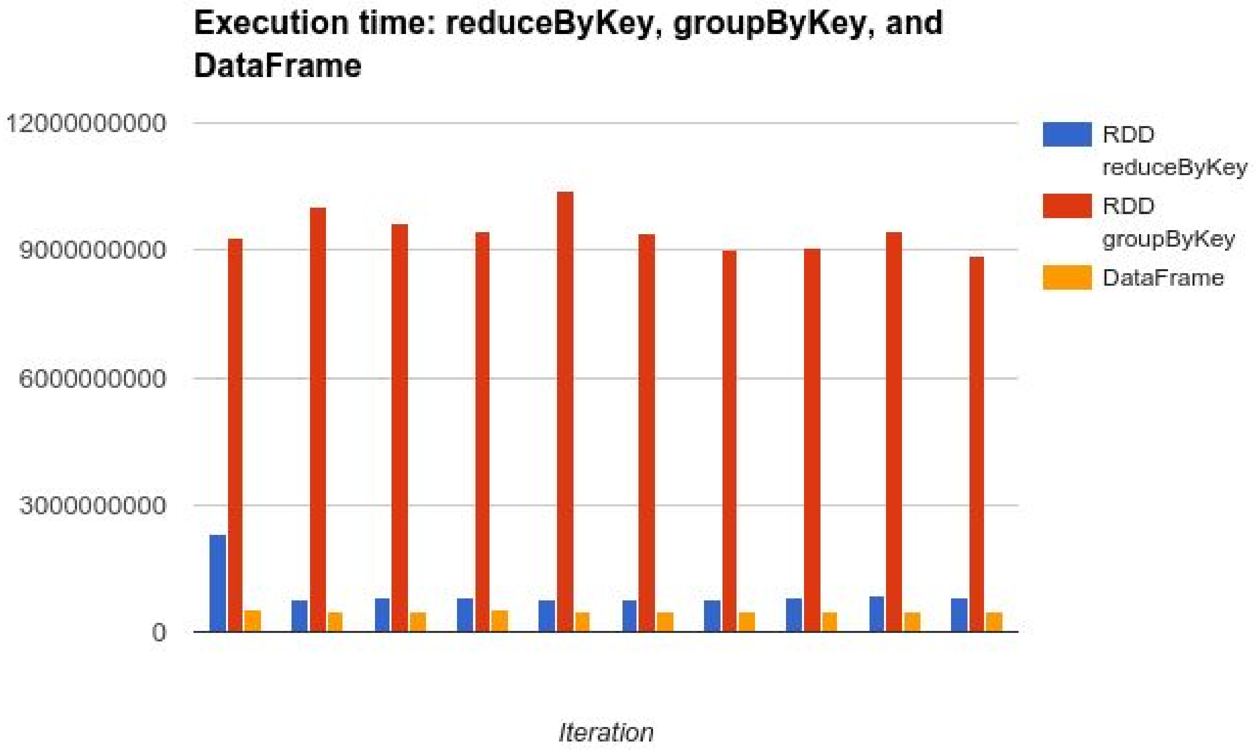
And functional style maps:

```
/**
 * Functional map + Dataset, sums the positive attributes for the
pandas
 */
def funMap(ds: Dataset[RawPanda]): Dataset[Double] = {
  ds.map{rp => rp.attributes.filter(_ > 0).sum}
}
```

Functional & Relational wordcount (in Python)

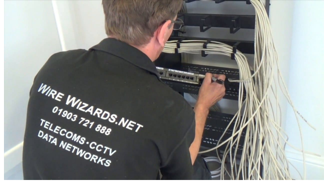
```python
# In Spark 2+ we need to convert to an RDD for functional queries
# Note: we could also do this by registering a UDF.
words = df.select("panda_name").rdd().flatMap(
    lambda row: row.panda_name.split(" "))

# Create a new DataFrame to count the number of words

words_df = words.map(lambda w: Row(word=w, cnt=1)).toDF()

word_counts = words_df.groupBy("word").sum()
```

How much faster can it be?



What can the optimizer do now?

- Sort on the serialized data
- Understand the aggregate (“partial aggregates”)
  - Could sort of do this before but not as awesomely, and only if we used reduceByKey - not groupByKey
- Pack bits nice and tight



What are relational transformers like?

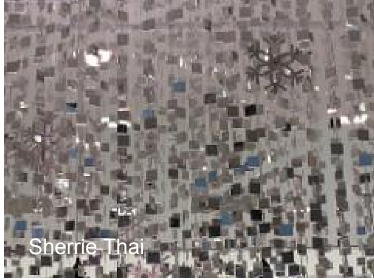
Many familiar faces are back with a twist:

- filter
- join
- groupBy - Now safe!

And some new ones:

- select
- window
- sql (register as a table and run “arbitrary” SQL)
- etc.

So whats this new groupBy?



- No longer causes explosions like RDD groupBy
  - Able to introspect and pipeline the aggregation
- Returns a GroupedData (or GroupedDataset)
- Makes it easy to perform multiple aggregations
- Built in shortcuts for aggregates like avg, min, max
- Longer list at http://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.sql.functions$
- Allows the optimizer to see what aggregates are being performed

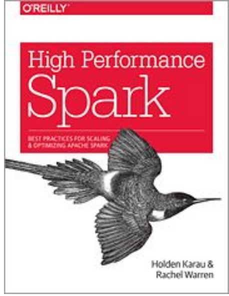
Easily compute multiple aggregates:


```
df.groupBy("age").agg(min("hours-per-week"),
                      avg("hours-per-week"),
                      max("capital-gain"))
```

But where Datasets explode?

- Iterative algorithms - large plans
- Some push downs are sad pandas :(
- Default shuffle size is sometimes too small for big data (200 partitions)
- Default partition size when reading in is also sad

High Performance Spark!



Focused on how to scale your Spark jobs

You can buy it from O'Reilly - http://bit.ly/highPerfSpark

Get notified when in print on Amazon:

- http://www.highperformancespark.com
- https://twitter.com/highperfspark

Where to go from here?

- Just getting started: Paco Nathan’s video
- Spark API docs
- Spark summit youtube videos (TheApacheSpark on YT)
- Spark & Everything (“Weekend Project”) @ 2:30 in Palais Atelier
- Reading the source code - not that bad?

k thnx bye!

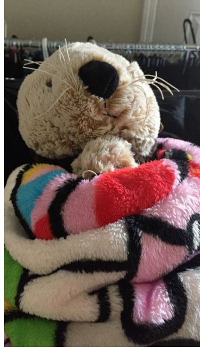
Will tweet results "eventually" @holdenkarau

If you care about Spark testing and don't hate surveys: http://bit.ly/holdenTestingSpark

PySpark Users: Have some simple UDFs you wish ran faster you are willing to share?: http://bit.ly/pySparkUDF

Pssst: Have feedback on the presentation? Give me a shout (holden@pigscanfly.ca) if you feel comfortable doing so :)

Our final bit of magic today (Python & co):



- Spark is written in Scala (runs on the JVM)
- Users want to work in their favourite language
- Python, R, C#, etc. all need a way to talk to the JVM
- How expensive could IPC be anyways? :P
- (Time Permitting)

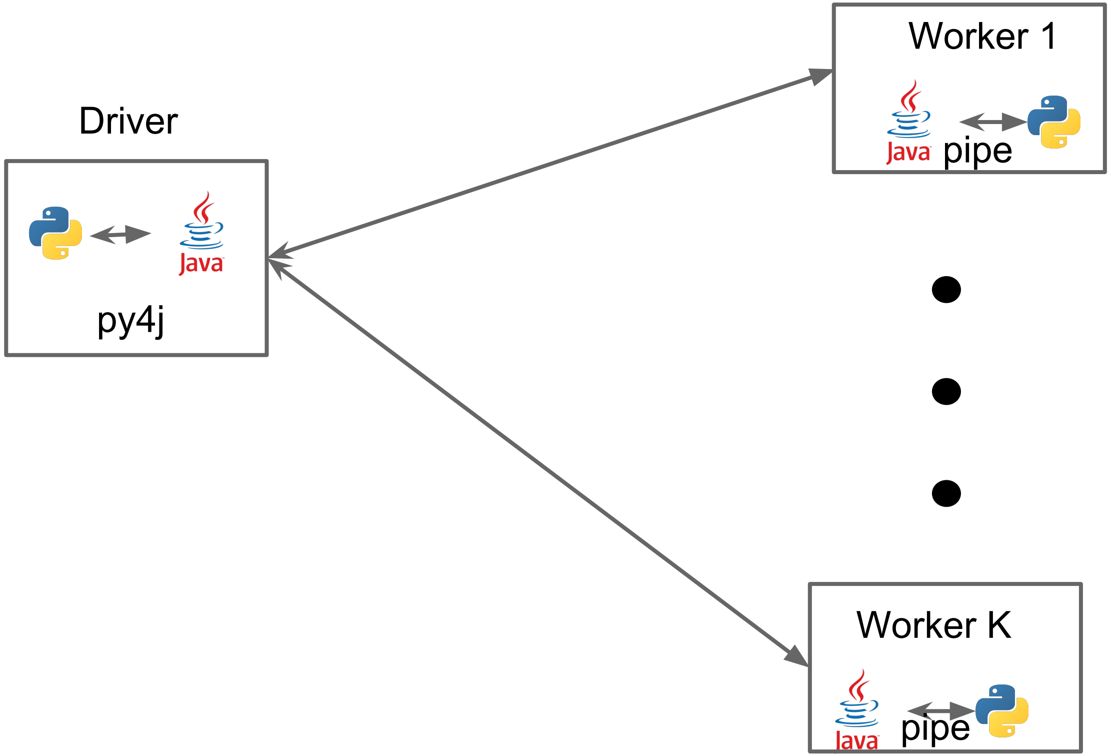
A (possible) quick detour into PySpark's

Photo by Bill Ward

Spark in Scala, how does PySpark work?

- Py4J + pickling + magic
  - This can be kind of slow sometimes
- RDDs are generally RDDs of pickled objects
- Spark SQL (and DataFrames) avoid some of this

So what does that look like?



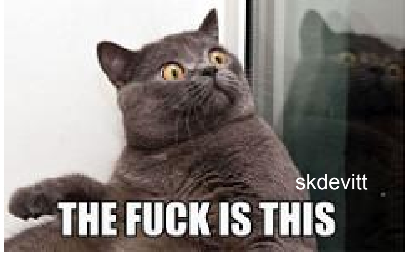
So how does this break?

- Data from Spark worker serialized and piped to Python worker
  - Multiple iterator-to-iterator transformations are still pipelined :)
- Double serialization cost makes everything more expensive
- Python worker startup takes a bit of extra time
- Python memory isn't controlled by the JVM - easy to go over container limits if deploying on YARN or similar
- etc.

What do the Python gnomes look like?

```
        self.is_cached = True
        javaStorageLevel =
self.ctx._getJavaStorageLevel(storageLevel)
        self._jrdd.persist(javaStorageLevel)
        return self
```

Spark specific terms in this talk

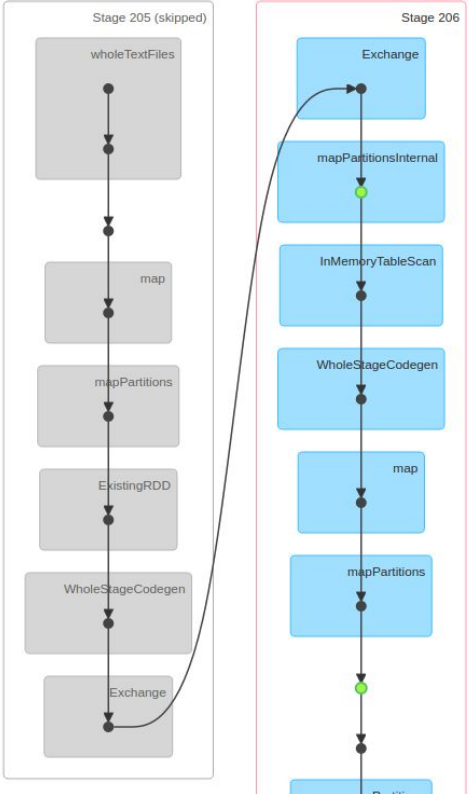


- RDD
  - Resilient Distributed Dataset - Like a distributed collection. Supports many of the same operations as Seq’s in Scala but automatically distributed and fault tolerant. Lazily evaluated, and handles faults by recompute. Any* Java or Kyro serializable object.
- DataFrame
  - Spark DataFrame - not a Pandas or R DataFrame. Distributed, supports a limited set of operations. Columnar structured, runtime schema information only. Limited* data types.
- Dataset
  - Compile time typed version of DataFrame (templated). **The future!** (not exclusively)

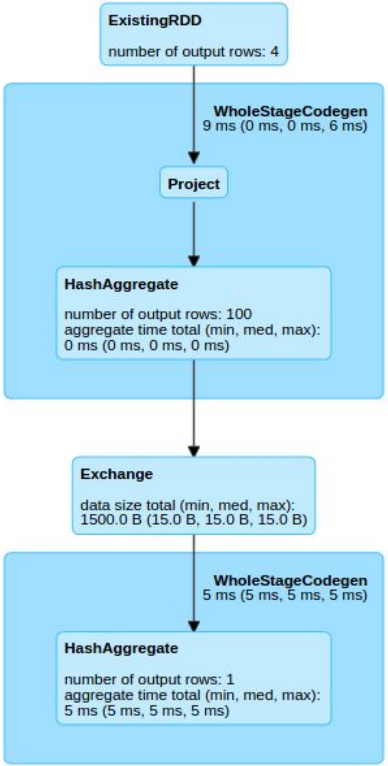
Magic part #1: the DAG

- In Spark most of our work is done by transformations
  - Things like map
- Transformations return new RDDs or DataFrames representing this data
- The RDD or DataFrame however doesn't really "exist"
- RDD & DataFrames are really just "plans" of how to make the data show up if we force Spark's hand
- tl;dr - the data doesn't exist until it "has" to

The DAG



The query plan



Word count (in python)


```
lines = sc.textFile(src)
words = lines.flatMap(lambda x: x.split(" "))
word_count =
  (words.map(lambda x: (x, 1))
             .reduceByKey(lambda x, y: x+y))
word_count.saveAsTextFile(“output”)
```

Word count (in python)

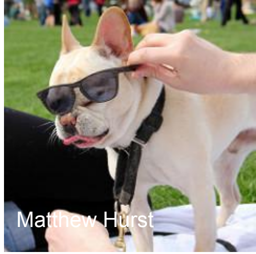

```
lines = sc.textFile(src)
words = lines.flatMap(lambda x: x.split(" "))
word_count =
  (words.map(lambda x: (x, 1))
          .reduceByKey(lambda x, y: x+y))
```

No data is read or processed until after this line

```
word_count.saveAsTextFile("output")
```

This is an "action" which forces spark to evaluate the RDD

How the DAG magic is awesome:

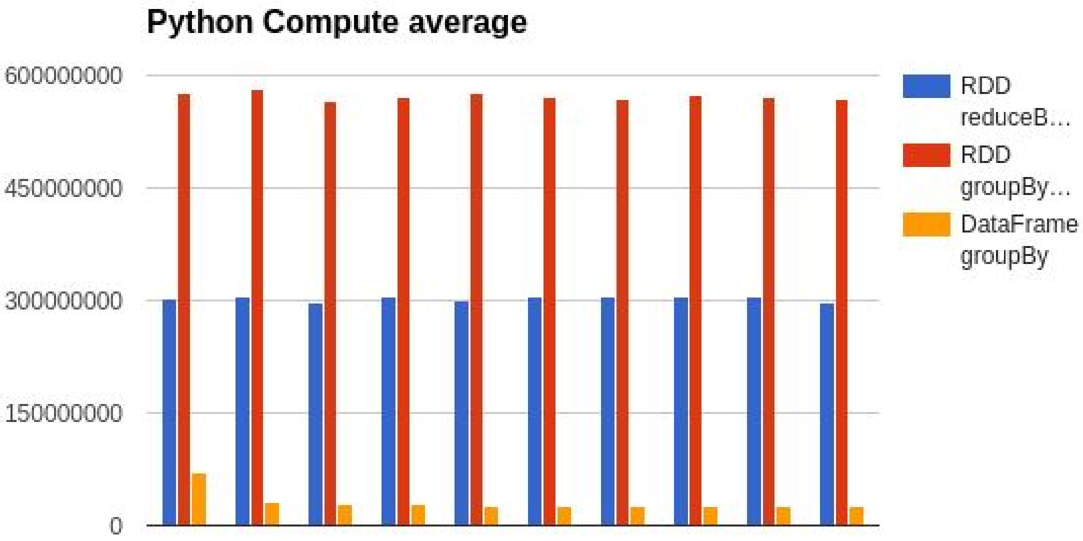


- Pipelining (can put maps, filter, flatMap together)
- Can do interesting optimizations by delaying work
- We use the DAG to recompute on failure
  - (writing data out to 3 disks on different machines is so last season)
  - Or the DAG puts the R is Resilient RDD, except DAG doesn’t have an R :(

And where it reaches its limits:

- It doesn’t have a whole program view
  - Can only see up to the action, can’t see into the next one
  - So we have to help Spark out and cache
- Combining the transformations together makes it hard to know what failed
- It can only see the pieces it understands
  - can see two maps but can’t tell what each map is doing

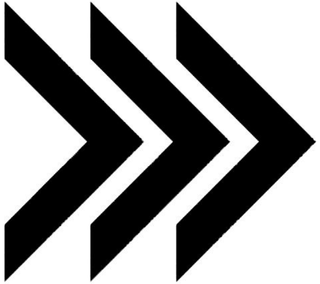
And back to magic with Dataframes:



*Note: do not compare absolute #s with previous graph - different dataset sizes because I forgot to write it down when I made the first one.

The “future*”: Faster interchange

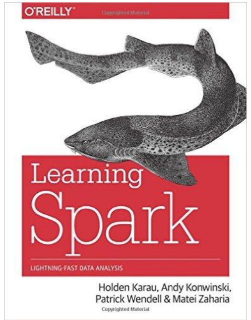
- Faster interchange between Python and Spark (e.g. Tungsten or Apache Arrow) (SPARK-13391 & SPARK-13534 + it’s PR)
- Willing to share your Python UDFs for benchmarking? - http://bit.ly/pySparkUDF



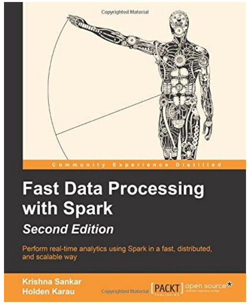
*The future may or may not have better performance than today. But bun-bun the bunny has some lettuce so its ok!

Spark Videos

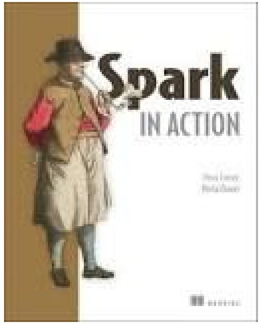
- Apache Spark Youtube Channel
- My Spark videos on YouTube -
    - http://bit.ly/holdenSparkVideos
- Spark Summit 2014 training
- Paco's Introduction to Apache Spark



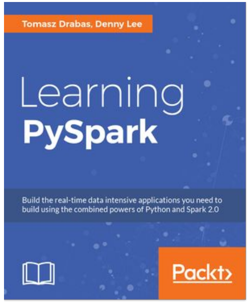
Learning Spark



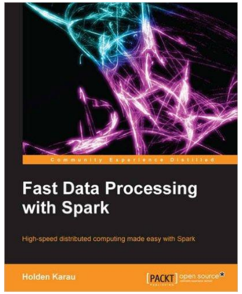
Fast Data Processing with Spark (2nd edition)



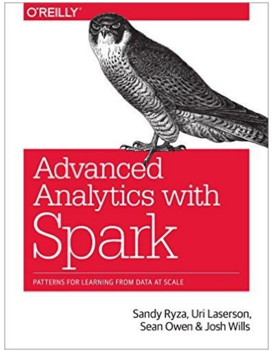
Spark in Action



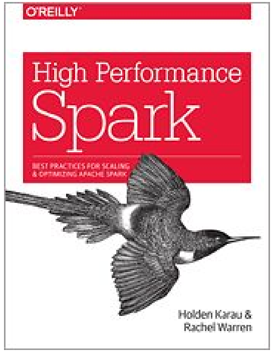
Learning PySpark



Fast Data Processing with Spark (Out of Date)

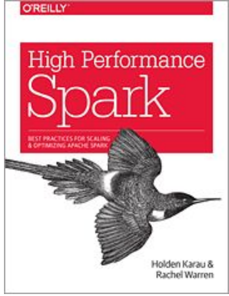


Advanced Analytics with Spark



High Performance Spark

High Performance Spark!



Available from O'Reilly TODAY - Amazon print "soon"

- Buy from O'Reilly - http://bit.ly/highPerfSpark

Get notified when in print on Amazon:

- http://www.highperformancespark.com
- https://twitter.com/highperfspark

* Early Release means extra mistakes, but also a chance to help us make a more awesome book.

And some upcoming talks:

- June
  - Berlin Buzzwords
  - Scala Swarm (Porto, Portugal)
- July
  - Scala Up North
- August
  - PyCon AU