Getting the best Performance with PySpark

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]
- previously Alpine, Databricks, Google, Foursquare & Amazon
- co-author of Learning Spark & Fast Data processing with Spark
  - co-author of a new book focused on Spark performance coming out next year*
- @holdenkarau
- Slide share http://www.slideshare.net/hkarau
- Linkedin https://www.linkedin.com/in/holdenkarau
- Github https://github.com/holdenk
- Spark Videos http://bit.ly/holdenSparkVideos

What is going to be covered:

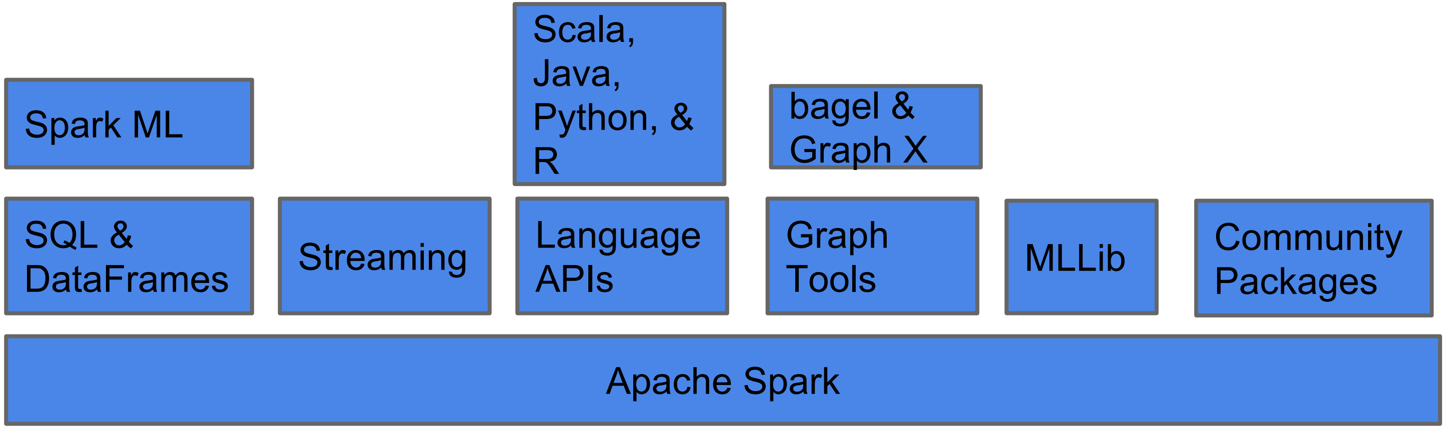
- What I think I might know about you
- A quick background of how PySpark works
- RDD re-use (caching, persistence levels, and checkpointing)
- Working with key/value data
  - Why group key is evil and what we can do about it
- When Spark SQL can be amazing and wonderful
- A brief introduction to Datasets (new in Spark 1.6)
- Calling Scala code from Python with Spark

Or….


Starring a 2AM page
You're process isn't there anymore
The OOM killer
You can't block the signal
Java Virtual Machine
R RESTRICTED
UNDER 17 REQUIRES ACCOMPANYING PARENT OR ADULT GUARDIAN
Huang
Yun
Chung


Starring Scala & Java
Supporting roles by Python and R
You're results have been collected
Happy Spark Land
I can't believe its not Map Reduce!
Unicorn by Celestial Dawn Comer
G GENERAL AUDIENCES
ALL AGES ADMITTED

The different pieces of Spark



Who I think you wonderful humans are?

- Nice* people
- Don’t mind pictures of cats
- Know some Apache Spark
- Want to scale your Apache Spark jobs
- Don’t overly mind a grab-bag of topics

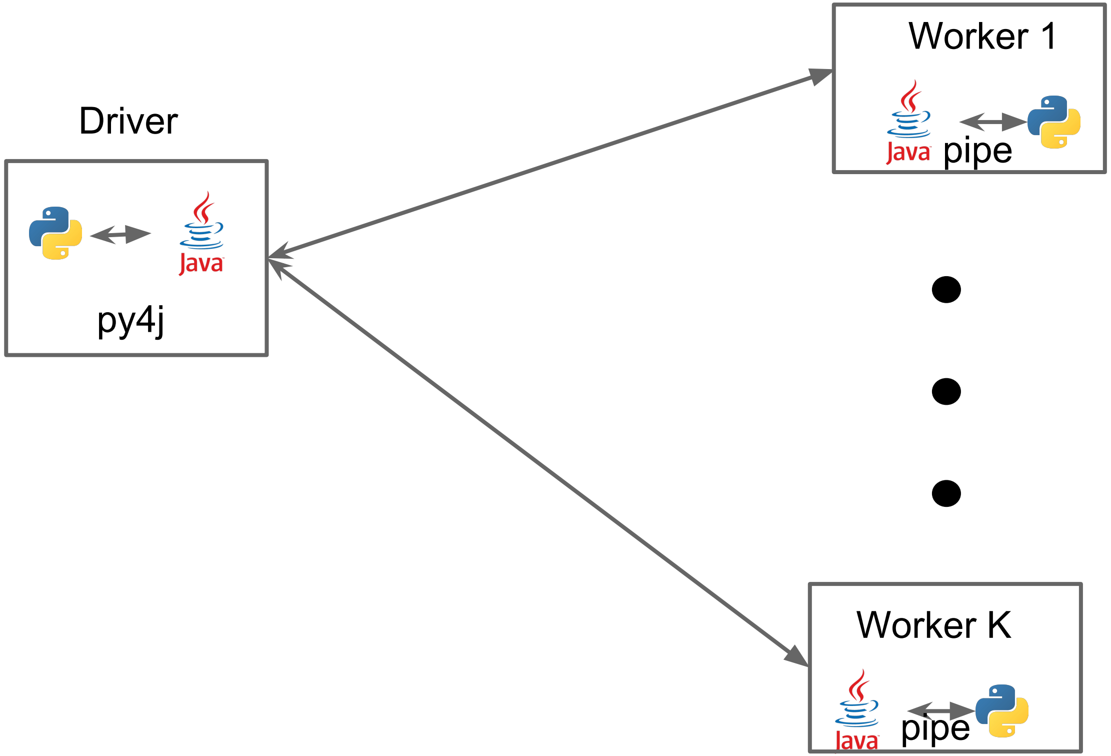
A detour into PySpark’s internals

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?

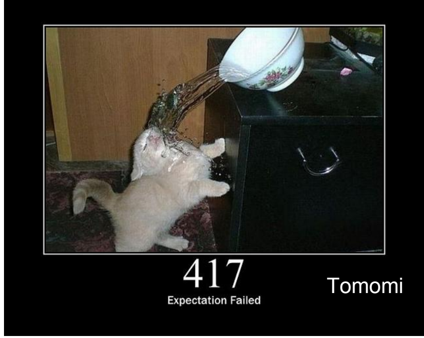


So how does that impact PySpark?

- 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
- Error messages make ~0 sense
- etc.

Photo from Cocoa Dream

Lets look at some old stand bys:




```
words = rdd.flatMap(lambda x: x.split(" "))
wordPairs = words.map(lambda w: (w, 1))
grouped = wordPairs.groupByKey()
grouped.mapValues(lambda counts: sum(counts))
warnings = rdd.filter(lambda x: x.lower.find("warning") != -1).
count()
```

RDD re-use - sadly not magic

- If we know we are going to re-use the RDD what should we do?
  - If it fits nicely in memory caching in memory
  - persisting at another level
    - MEMORY, MEMORY_AND_DISK
  - checkpointing
- Noisey clusters
  - _2 & checkpointing can help
- persist first for checkpointing

What is key skew and why do we care?

- 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)

groupByKey - just how evil is it?

- Pretty evil
- Groups all of the records with the same key into a single record
  - Even if we immediately reduce it (e.g. sum it or similar)
  - This can be too big to fit in memory, then our job fails
- Unless we are in SQL then happy pandas

So what does that look like?

(94110, A, B)
(94110, A, C)
(10003, D, E)
(94110, E, F)

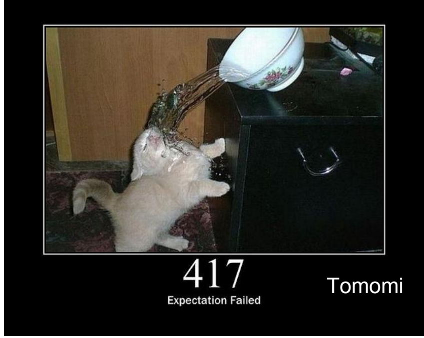
(94110, A, R)
(10003, A, R)
(94110, D, R)
(94110, E, R)

(94110, E, R)
(67843, T, R)
(94110, T, R)
(94110, T, R)

(94110, [(A, B), (A, C), (E, F), (A, R), (D, R), (E, R), (E, R), (T, R) (T, R)]

(10003, A, R)

(67843, T, R)



“Normal” Word count w/RDDs

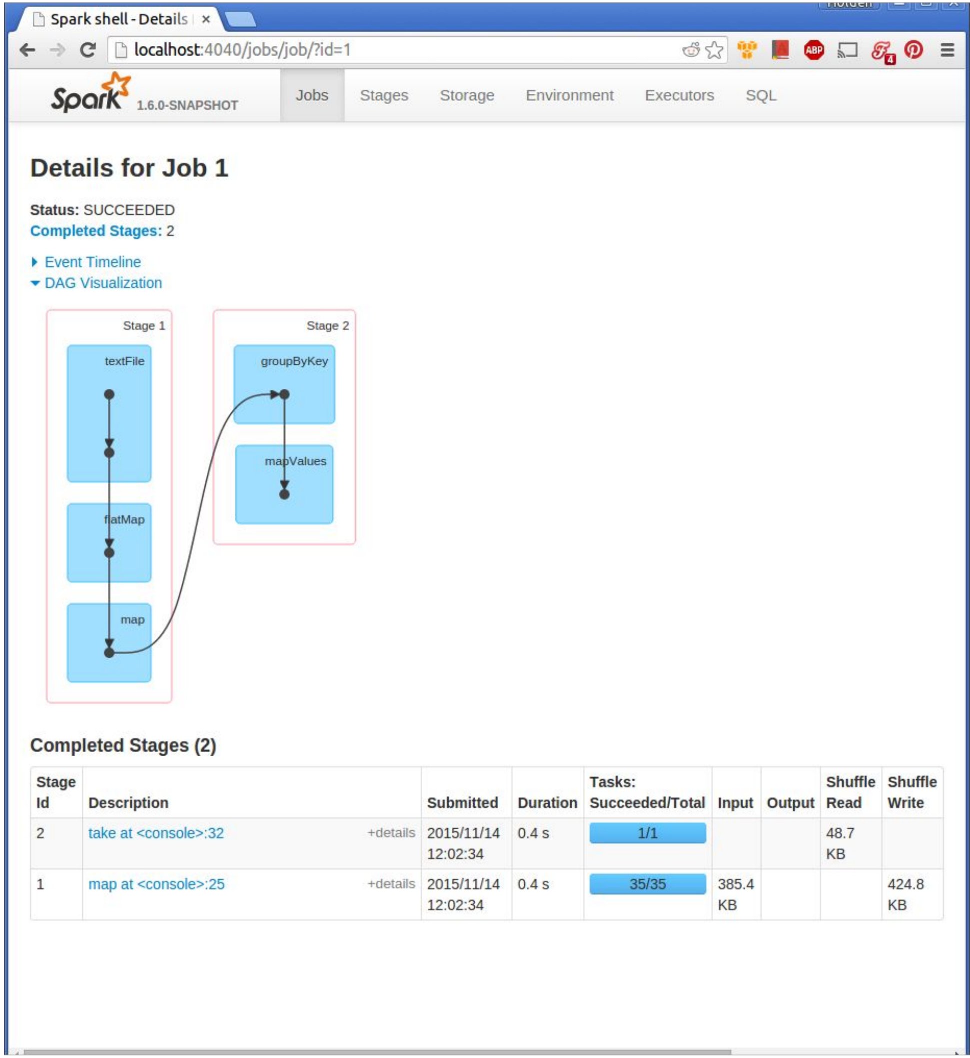
```
lines = sc.textFile(src)
words = lines.flatMap(lambda x: x.split(" "))
word_count =
  (words.map(lambda x: (x, 1))
          .reduceByKey(lambda x, y: x+y))
word_count.saveAsTextFile(output)
```

These are still pipelined inside of the same python executor

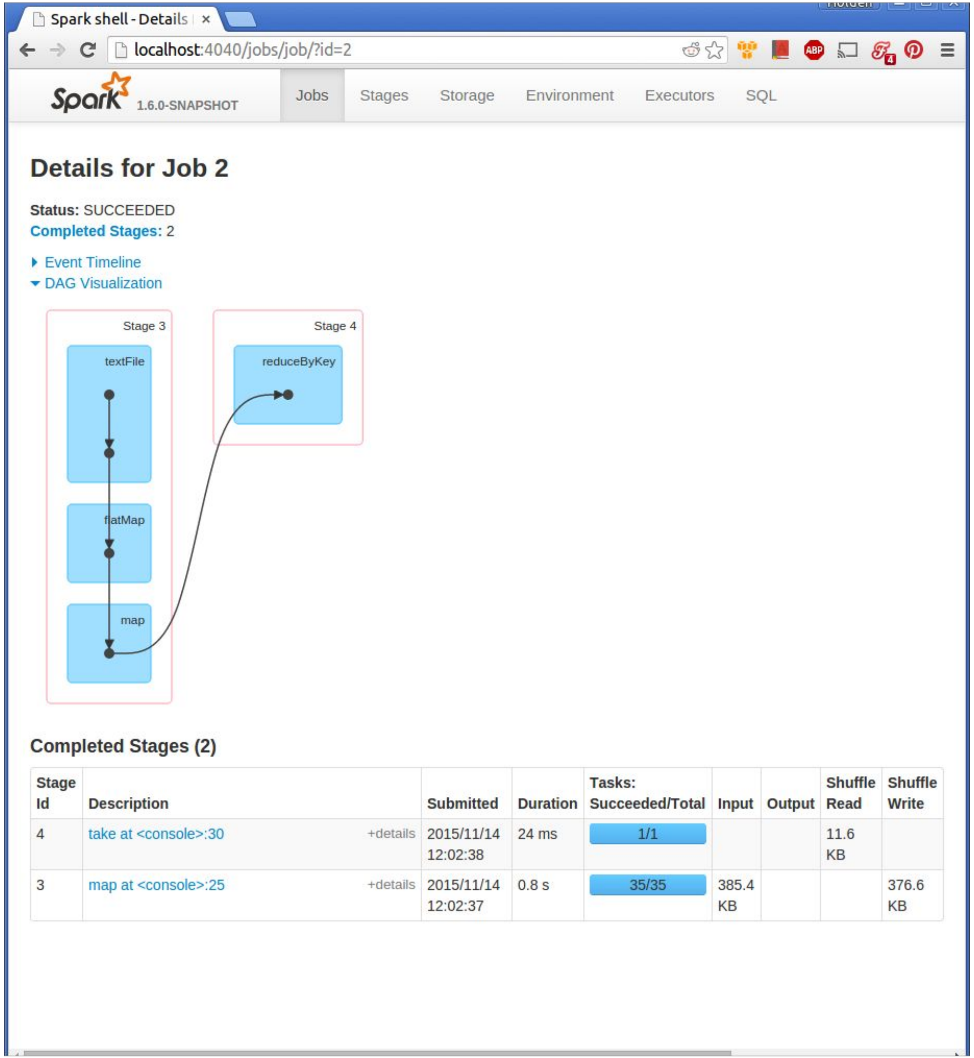
No data is read or processed until after this line

This is an “action” which forces spark to evaluate the RDD

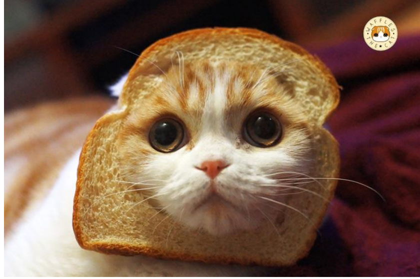
GroupByKey


Spark shell - Details
localhost:4040/jobs/job/?id=1
Spark 1.6.0-SNAPSHOT
Jobs
Stages
Storage
Environment
Executors
SQL
Details for Job 1
Status: SUCCEEDED
Completed Stages: 2
Event Timeline
DAG Visualization
Stage 1
textFile
flatMap
map
Stage 2
groupByKey
mapValues
Completed Stages (2)
Stage Id
Description
Submitted
Duration
Tasks: Succeeded/Total
Input
Output
Shuffle Read
Shuffle Write
2
take at <console>:32
+details
2015/11/14 12:02:34
0.4 s
1/1
48.7 KB
1
map at <console>:25
+details
2015/11/14 12:02:34
0.4 s
35/35
385.4 KB
424.8 KB

reduceByKey



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)

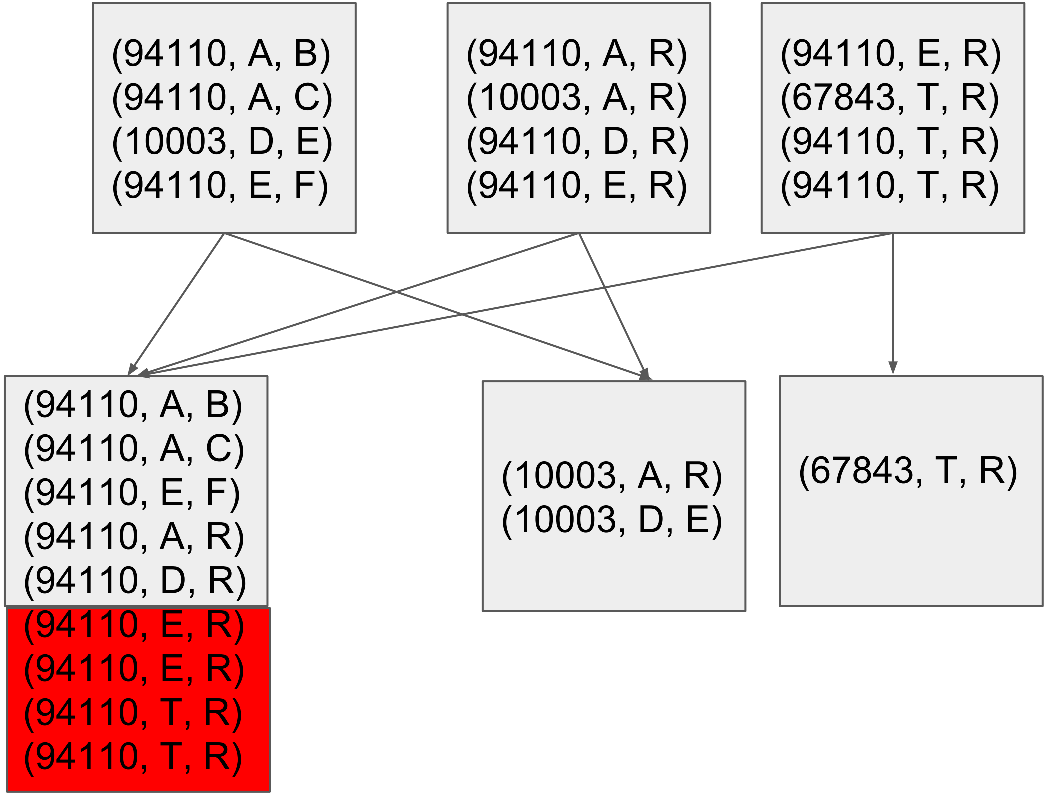
Can just the shuffle cause problems?

- Sorting by key can put all of the records in the same partition
- We can run into partition size limits (around 2GB)
- Or just get bad performance

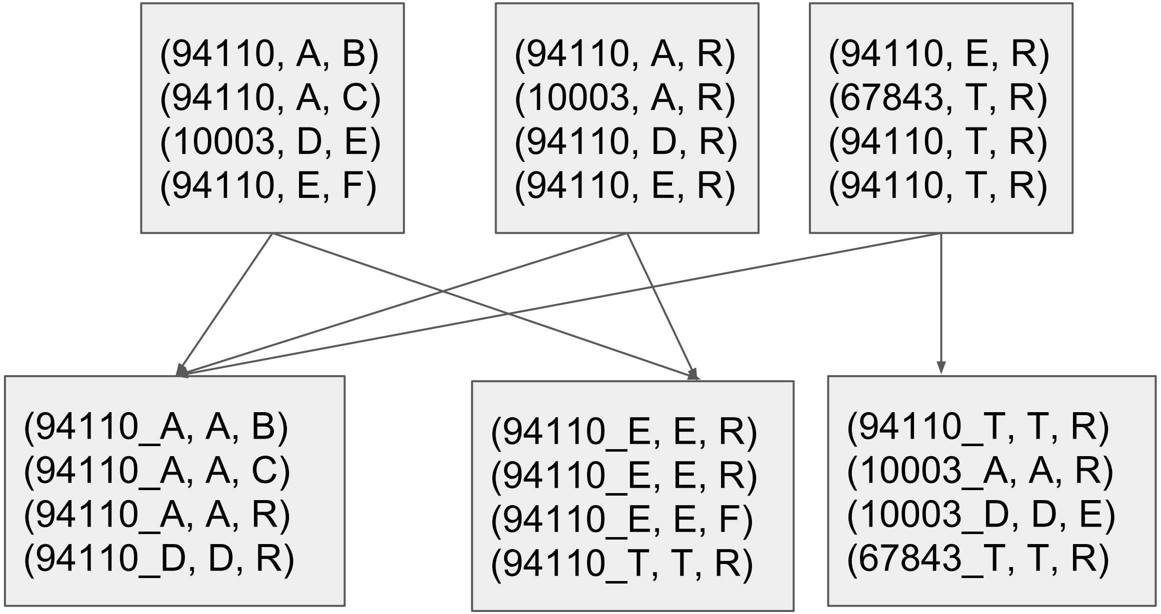
(94110, A, B) (94110, A, C) (10003, D, E) (94110, E, F)	(94110, A, R) (10003, A, R) (94110, D, R) (94110, E, R)	(94110, E, R) (67843, T, R) (94110, T, R) (94110, T, R)

- So we can handle data like the above we can add some “junk” to our key

Shuffle explosions :(



100% less explosions



Well there is a bit of magic in the shuffle….

- We can reuse shuffle files
- But it can (and does) explode*

Sculpture by Flaming Lotus Girls
Photo by Zaskoda

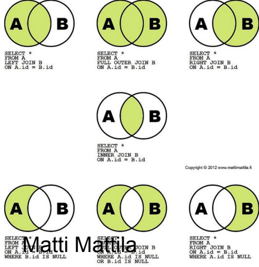
Our saviour from serialization: DataFrames

- For the most part keeps data in the JVM
  - Notable exception is UDFs written in Python
- Takes our python calls and turns it into a query plan
- If we need more than the native operations in Spark's DataFrames
- be wary of Distributed Systems bringing claims of usability….

So what are Spark DataFrames?

- More than SQL tables
- Not Pandas or R DataFrames
- Semi-structured (have schema information)
- tabular
- work on expression instead of lambdas
    - e.g. df.filter(df.col("happy") == true) instead of rdd.filter(lambda x: x.happy == true))

Where can Spark SQL benefit perf?

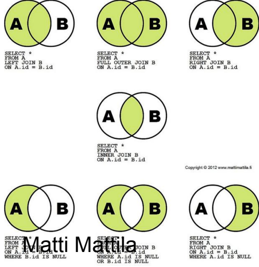


Matti Mattila

- Structured or semi-structured data
- OK with having less* complex operations available to us
- We may only need to operate on a subset of the data
  - The fastest data to process isn't even read
- Remember that non-magic cat? Its got some magic** now
  - In part from peeking inside of boxes
- non-JVM (aka Python & R) users: saved from double serialization cost! :)

**Magic may cause stack overflow. Not valid in all states. Consult local magic bureau before attempting magic

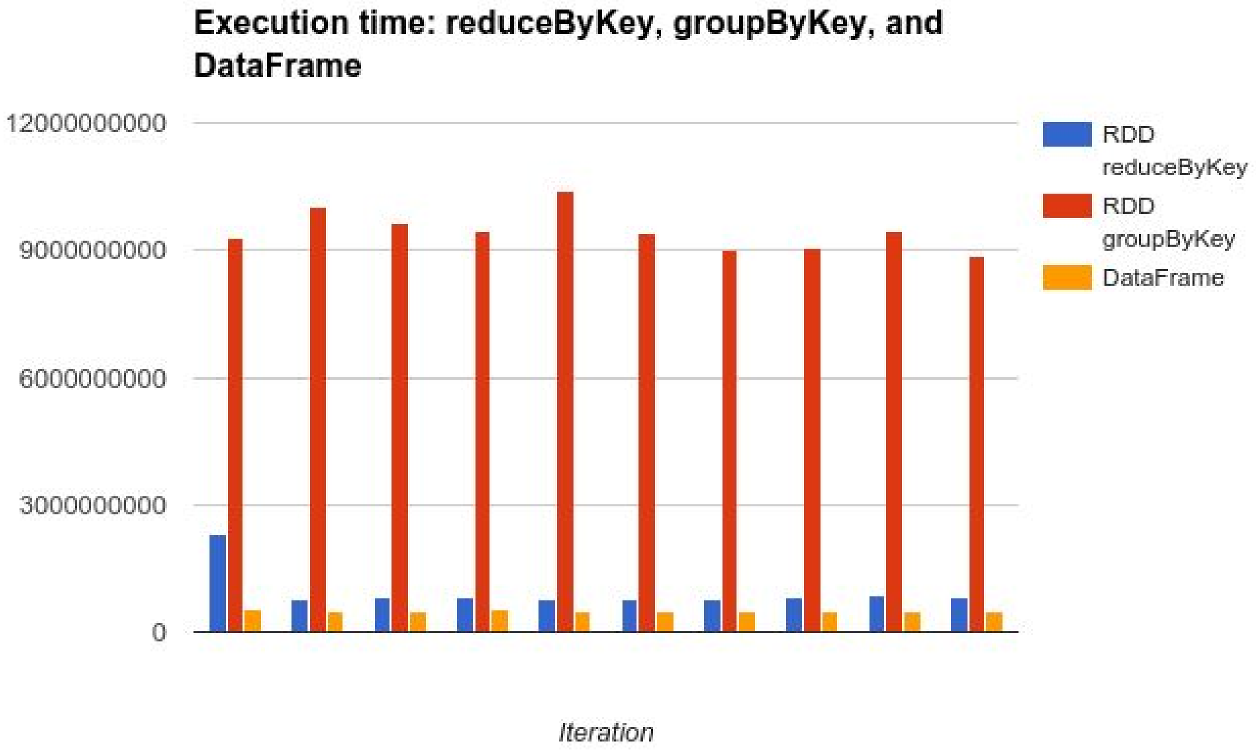
Why is Spark SQL good for those things?



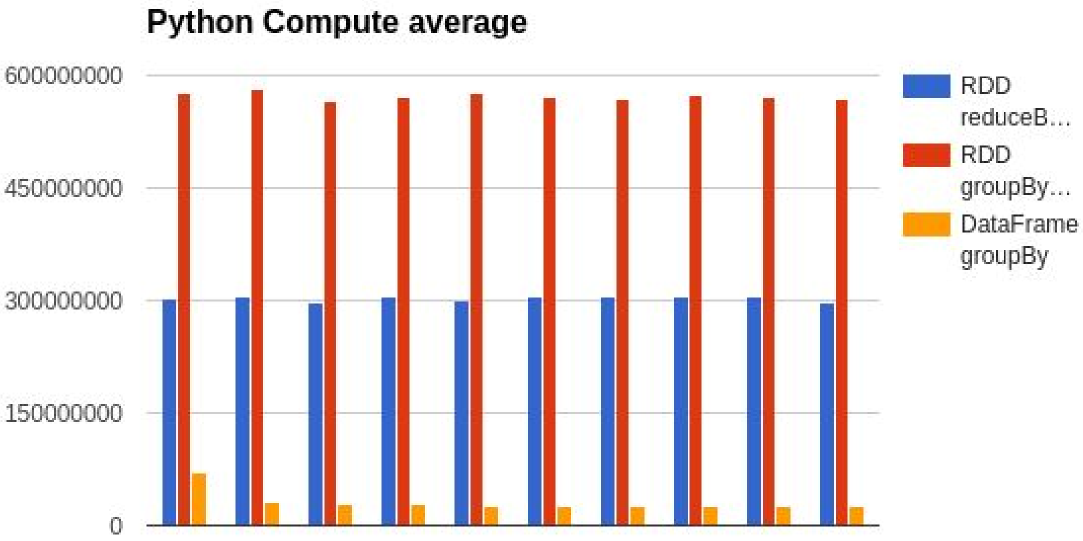
Matti Mattila

- Space efficient columnar cached representation
- Able to push down operations to the data store
- Optimizer is able to look inside of our operations
    - Regular spark can't see inside our operations to spot the difference between (min(_, _)) and (append(_, _))

How much faster can it be? (Scala)



How much faster can it be? (Python)



*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.

Word count w/Dataframes

Still have the double serialization here :(

```
df = sqlCtx.read.load(src)
# Returns an RDD
words = df.select("text").flatMap(lambda x: x.text.split(" "))
words_df = words.map(
  lambda x: Row(word=x, cnt=1)).toDF()
word_count = words_df.groupBy("word").sum()
word_count.write.format("parquet").save("wc.parquet")
```

Or we can make a UDF

```
def function(x):
    # Some magic
sqlContext.registerFunction("name", function,
IntegerType())
```

Buuuut….

- Our UDFs will be “slow” (e.g. require data copy from executor and back)

Mixing Python & JVM code FTW:

- DataFrames are an example of pushing our processing to the JVM
- Python UDFS & maps lose this benefit
- But we can write Scala UDFS and call them from Python
  - py4j error messages can be difficult to understand :(
- Trickier with RDDs since stores pickled objects

Exposing functions to be callable from Python:

```
// functions we want to be callable from python
object functions {
  def kurtosis(e: Column): Column = new Column
(Kurtosis(EvilSqlTools.getExpr(e)))
  def registerUdfs(sqlCtx: SQLContext): Unit = {
    sqlCtx.udf.register("rowKurtosis", helpers.rowKurtosis _)
  }
}
```

Calling the functions with py4j*:

- The SparkContext has a reference to the jvm (_jvm)
- Many Python objects which are wrappers of JVM objects have _j[objtype] to get the JVM object
    - rdd._jrdd
    - df._jdf
    - sc._jsc
- These are all private and may change

*The py4j bridge only exists on the driver**
** Not exactly true but close enough

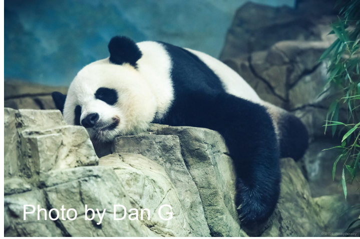
e.g.:

```python
def register_sql_extensions(sql_ctx):
    scala_sql_context = sql_ctx._ssql_ctx
    spark_ctx = sql_ctx._sc
    (spark_ctx._jvm.com.sparklingpandas.functions
     .registerUdfs(scala_sql_context))
```

More things to keep in mind with DFs (in Python)

- Schema serialized as json from JVM
- toPandas is essentially collect
- joins can result in the cross product
  - big data x big data =~ out of memory
- Pre 2.0: Use the HiveContext
  - you don’t need a hive install
  - more powerful UDFs, window functions, etc.

DataFrames aren’t quite as lazy...

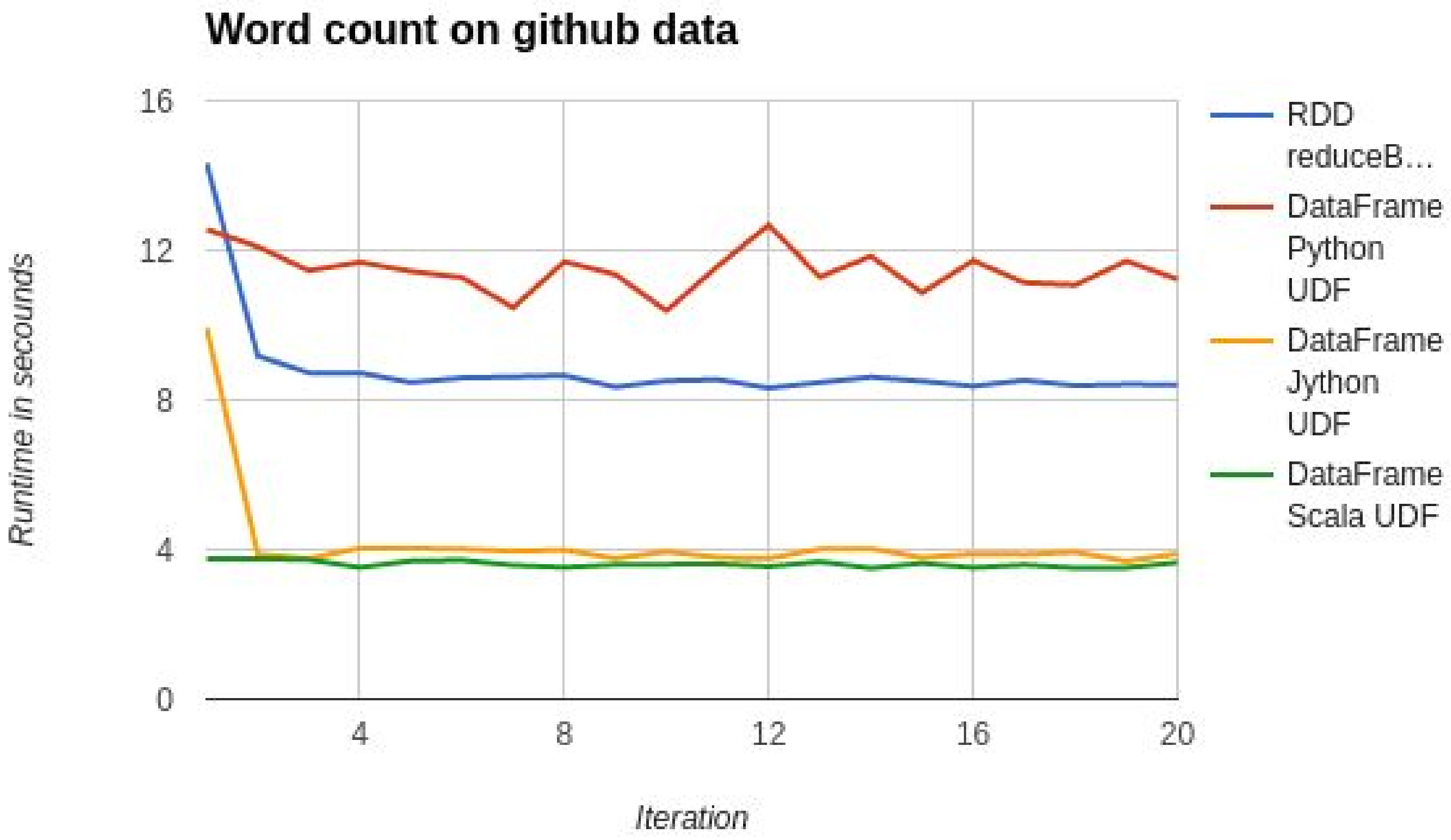


- Keep track of schema information
- Loading JSON data involves looking at the data
- Before if we tried to load non-existent data wouldn’t fail right away, now fails right away

The "future*": Awesome UDFs

- Work going on in Scala land to translate simple Scala into SQL expressions - need the Dataset API
  - Maybe we can try similar approaches with Python?
- Very early work going on to use Jython for simple UDFs (e.g. 2.7 compat & no native libraries) - SPARK-15369
  - Early benchmarking w/word count 5% slower than native Scala UDF, close to 65% faster than regular Python
- 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!



Word count on github data
16
12
8
4
0
4
8
12
16
20
Runtime in secounds
Iteration
RDD reduceB...
DataFrame Python UDF
DataFrame Jython UDF
DataFrame Scala UDF

The “future*”: Faster interchange

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

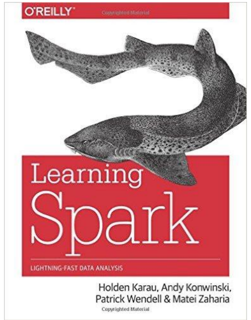
*The future may or may not have better performance than today. But bun-bun the bunny has some lettuce so its ok!

Spark Testing Resources

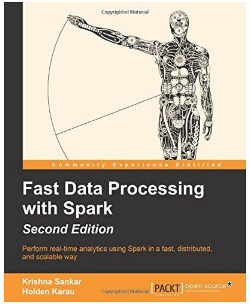
- Libraries
  - Scala: spark-testing-base (scalacheck & unit) sscheck (scalacheck) example-spark (unit)
  - Java: spark-testing-base (unit)
  - Python: spark-testing-base (unittest2), pyspark.test (pytest)
- Strata San Jose Talk (up on YouTube)
- Blog posts
  - Unit Testing Spark with Java by Jesse Anderson
  - Making Apache Spark Testing Easy with Spark Testing Base
  - Unit testing Apache Spark with py.test

Additional Spark Resources

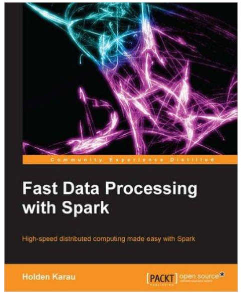
- Programming guide (along with JavaDoc, PyDoc, ScalaDoc, etc.)
  - http://spark.apache.org/docs/latest/
- Kay Ousterhout's work
  - http://www.eecs.berkeley.edu/~keo/
- Books
- Videos
- Spark Office Hours
  - Normally in the bay area - will do Google Hangouts ones soon
  - follow me on twitter for future ones - https://twitter.com/holdenkarau



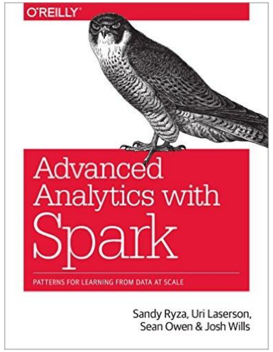
Learning Spark



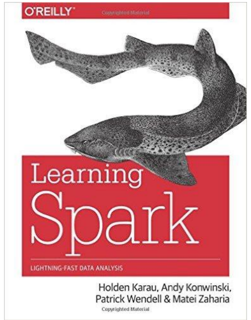
Fast Data
Processing with
Spark
(2nd edition)



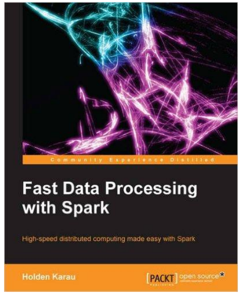
Fast Data
Processing with
Spark
(Out of Date)



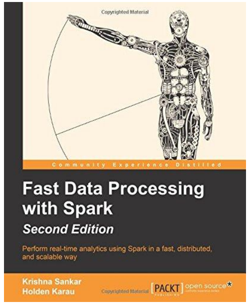
Advanced
Analytics with
Spark



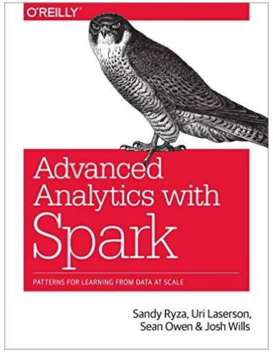
Learning Spark



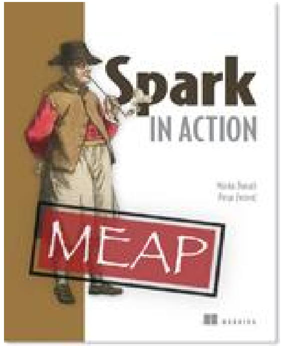
Fast Data
Processing with
Spark
(Out of Date)



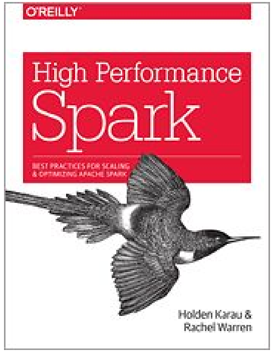
Fast Data
Processing with
Spark
(2nd edition)



Advanced
Analytics with
Spark

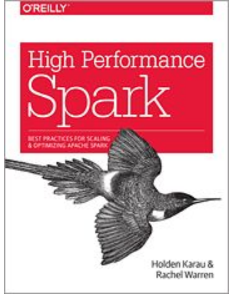


Coming soon:
Spark in Action



Coming soon:
High Performance Spark

And the next book…..



First four chapters are available in “Early Release”*:

- Buy from O’Reilly - http://bit.ly/highPerfSpark
- Chapter 9(ish) - Going Beyond Scala

Get notified when updated & finished:

- 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.

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

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