USING APACHE SPARK FOR ANALYTICS IN THE CLOUD

William C. Benton

Principal Software Engineer

Red Hat Emerging Technology

June 24, 2015

RED HAT SUMMIT

ABOUT ME

- Distributed systems and data science in Red Hat's Emerging Technology group
- Active open-source and Fedora developer
- Before Red Hat: programming language research

FORECAST

- Distributed data processing: history and mythology
- Data processing in the cloud
- Introducing Apache Spark
- How we use Spark for data science at Red Hat

DATA PROCESSING

Recent history & persistent mythology

CHALLENGES

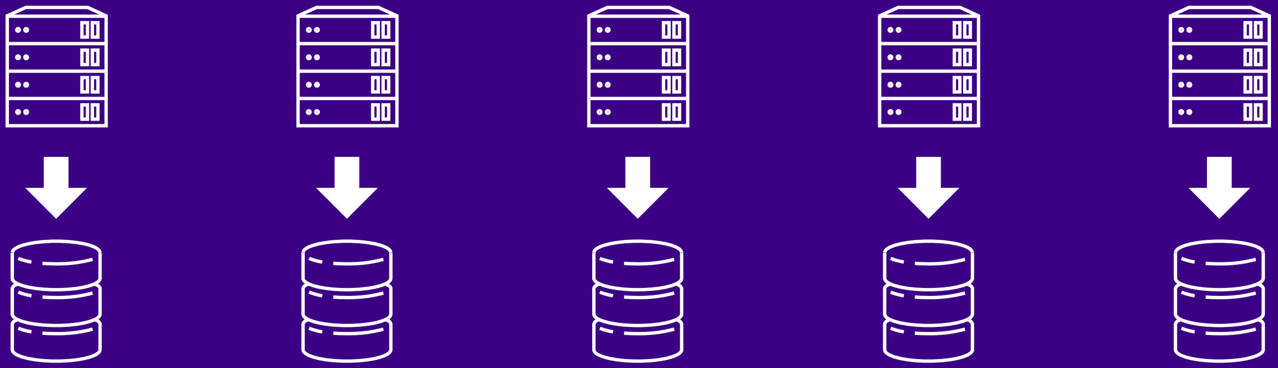
What makes distributed data processing difficult?

MAPREDUCE (2004)

- A novel application of some very old functional programming ideas to distributed computing
- All data are modeled as key-value pairs
- *Mappers* transform pairs; *reducers* merge several pairs with the same key into one new pair
- Runtime system *shuffles* data to improve locality

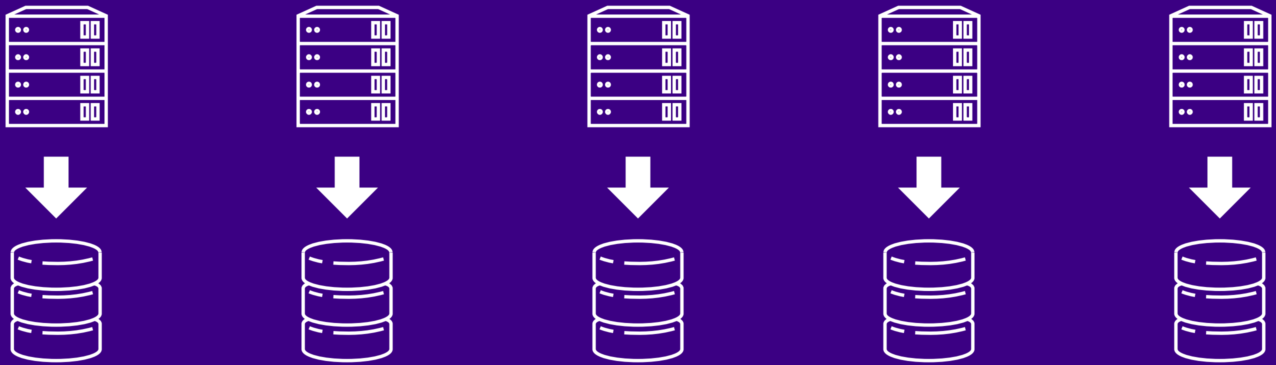
WORD COUNT

`"a b"` `"c e"` `"a b"` `"d a"` `"d b"`



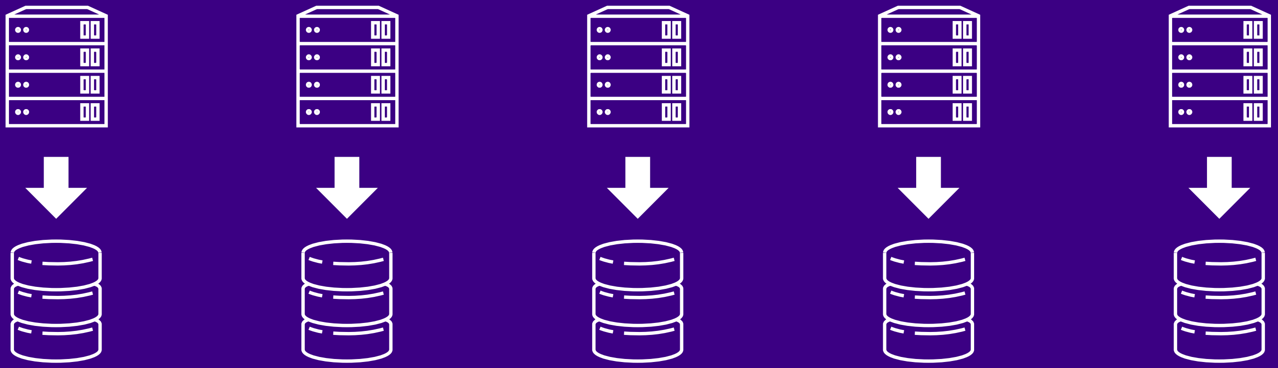
MAPPED INPUTS

```
(a, 1)  (c, 1)  (a, 1)  (d, 1)  (d, 1)
(b, 1)  (e, 1)  (b, 1)  (a, 1)  (b, 1)
```

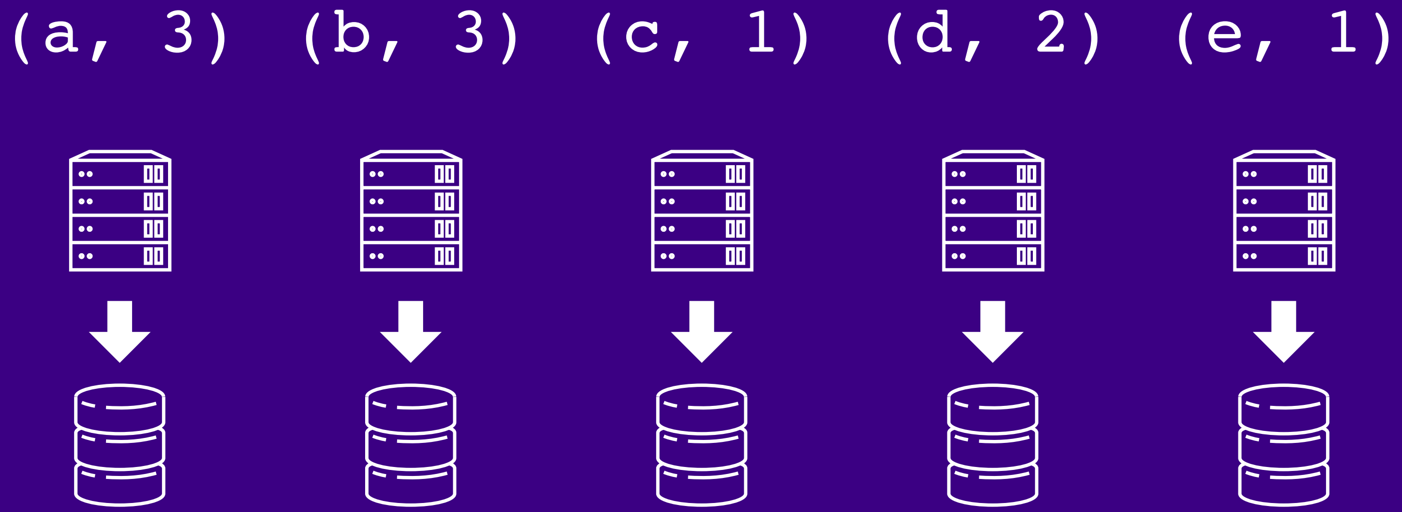


SHUFFLED RECORDS

(a, 1)	(b, 1)	(c, 1)	(d, 1)	(e, 1)
(a, 1)	(b, 1)		(d, 1)	
(a, 1)	(b, 1)			



REDUCED RECORDS



HADOOP (2005)

- Open-source implementation of MapReduce, a distributed filesystem, and more
- Inexpensive way to store and process data with scale-out on commodity hardware
- Motivates many of the default assumptions we make about “big data” today

“FACTS”

- You need an architecture that will scale out to many nodes to handle real-world data analytics
- Your network and disks probably aren't fast enough
- Locality is everything: you *need* to be able to run compute jobs on the nodes storing your data

“FACTS”

- You need an architecture that will scale out to many nodes to handle real-world data analytics
- Your network and disks probably aren't fast enough
- Locality is everything: you *need* to be able to run compute jobs on the nodes storing your data

> …at least two analytics production clusters (at Microsoft and Yahoo) have median job input sizes under 14 GB and 90% of jobs on a Facebook cluster have input sizes under 100 GB.

Appuswamy et al., “Nobody ever got fired for buying a cluster.” Microsoft Research Tech Report.

Takeaway #1: you may need *petascale* storage, but you probably don't even need *terascale* compute.

Takeaway #2: moderately sized workloads benefit more from *scale-up* than *scale out*.

“FACTS”

- You need an architecture that will scale out to many nodes to handle real-world data analytics
- Your network and disks probably aren't fast enough
- Locality is everything: you *need* to be able to run compute jobs on the nodes storing your data

Contrary to our expectations … CPU (and not I/O) is often the bottleneck [and] improving network performance can improve job completion time by a median of at most 2%

Ousterhout et al., “Making Sense of Performance in Data Analytics Frameworks.” *USENIX NSDI ’15*.

Takeaway #3: I/O is not the bottleneck (especially in moderately-sized jobs); focus on CPU performance.

“FACTS”

- You need an architecture that will scale out to many nodes to handle real-world data analytics
- Your network and disks probably aren't fast enough
- Locality is everything: you *need* to be able to run compute jobs on the nodes storing your data

Takeaway #4: collocated data and compute was a sensible choice for petascale jobs in 2005, but shouldn't necessarily be the default today.

FACTS (REVISED)

- You probably don't need an architecture that will scale out to many nodes to handle real-world data analytics (and might be better served by scaling up)
- Your network and disks probably aren't the problem
- You have enormous flexibility to choose the best technologies for storage and compute

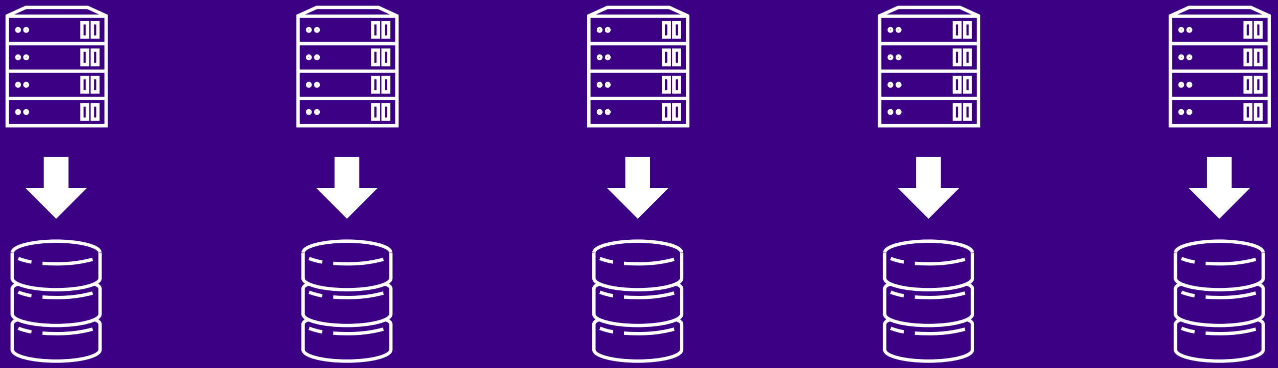
HADOOP IN 2015

- MapReduce is low-level, verbose, and not an obvious fit for many interesting problems
- No unified abstractions: Hive or Pig for query, Giraph for graph, Mahout for machine learning, etc.
- Fundamental architectural assumptions need to be revisited along with the “facts” motivating them

DATA PROCESSING IN THE CLOUD

How our assumptions should change

COLLOCATED DATA AND COMPUTE



ELASTIC RESOURCES

DISTINCT STORAGE AND COMPUTE

Combine the best storage system for your application with elastic compute resources.


INTRODUCING SPARK

Apache Spark is a framework for distributed computing based on a high-level, expressive abstraction.


Query
ML
Graph
Streaming
Spark core


Query
ML
Graph
Streaming
Spark core

Language bindings for Scala,
Java, Python, and R

Access data from JDBC,
Gluster, HDFS, S3, and more

A resilient distributed dataset is a partitioned, immutable, lazy collection.

ad hoc	Mesos	YARN

A resilient distributed dataset is a partitioned, immutable, lazy collection.

A resilient distributed dataset is a partitioned, immutable, lazy collection.

The PARTITIONS making up an RDD can be distributed across multiple machines

A resilient distributed dataset is a partitioned, immutable, lazy collection.

TRANSFORMATIONS create new (lazy) collections; ACTIONS force computations and return results

CREATING AN RDD

```
# from an in-memory collection
spark.parallelize(range(1, 1000))

# from the lines of a text file
spark.textFile("hamlet.txt")

# from a Hadoop-format binary file
spark.hadoopFile("...")
spark.sequenceFile("...")
spark.objectFile("...")
```

TRANSFORMING RDDS

```
# transform each element independently
numbers.map(lambda x: x + 1)

# turn each element into zero or more elements
lines.flatMap(lambda s: s.split(" "))

# reject elements that don't satisfy a predicate
vowels = ['a', 'e', 'i', 'o', 'u']
words.filter(lambda s: s[0] in vowels)

# keep only one copy of duplicate elements
words.distinct()
```

TRANSFORMING RDDS

```
# return an RDD of key-value pairs, sorted by
# the keys of each
pairs.sortByKey()

# combine every two pairs having the same key,
# using the given reduce function
pairs.reduceByKey(lambda x, y: max(x, y))

# join together two RDDs of pairs so that
# [(a, b)] join [(a, c)] == [(a, (b, c))]
pairs.join(other_pairs)
```

CACHING RESULTS

```
# tell Spark to cache this RDD in cluster
# memory after we compute it
sorted_pairs = pairs.sortByKey()
sorted_pairs.cache()


# as above, except also store a copy on disk
sorted_pairs.persist(MEMORY_AND_DISK)

# uncache and free this result
sorted_pairs.unpersist()
```

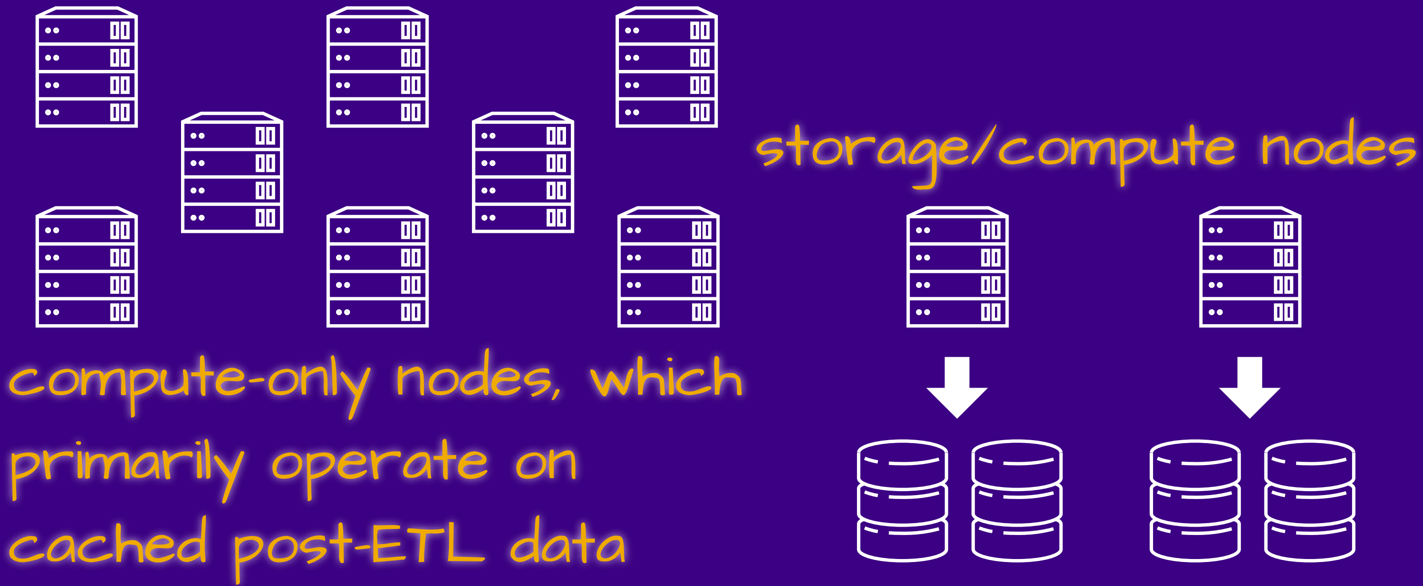
COMPUTING RESULTS

```
# compute this RDD and return a
# count of elements
numbers.count()

# compute this RDD and materialize it
# as a local collection
counts.collect()

# compute this RDD and write each
# partition to stable storage
words.saveAsTextFile("...")
```

WORD COUNT EXAMPLE

```
# create an RDD backed by the lines of a file
f = spark.textFile("...")
# ...mapping from lines of text to words
words = f.flatMap(lambda line: line.split(" "))
# ...mapping from words to occurrences
occs = words.map(lambda word: (word, 1))
# ...reducing occurrences to counts
counts = occs.reduceByKey(lambda a, b: a + b)
# POP QUIZ: what have we computed so far?
counts.saveAsTextFile("...")
```

PETASCALE STORAGE, IN-MEMORY COMPUTE



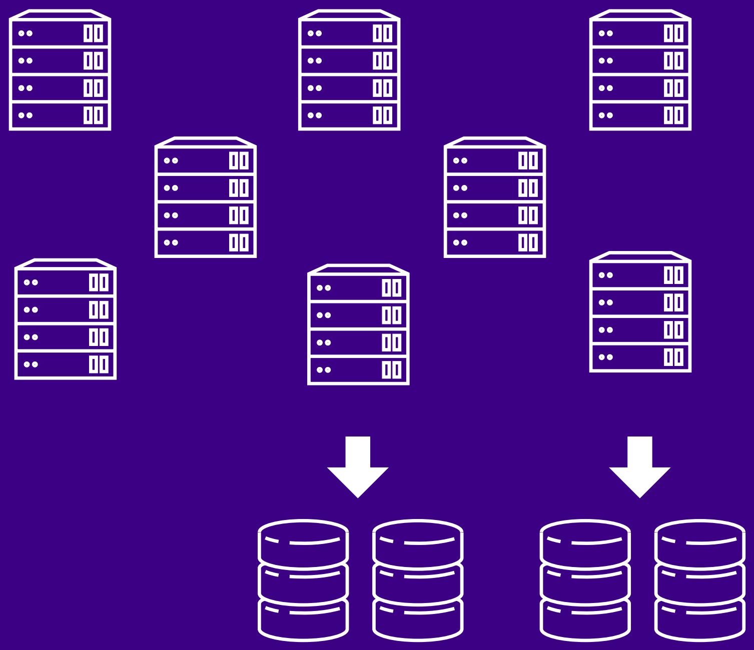
DATA SCIENCE AT RED HAT

THE EMERGING TECH DATA SCIENCE TEAM

- Engineers with distributed systems, data science, and scientific computing expertise
- Goal: help internal customers solve data problems and make data-driven decisions
- Principles: identify best practices, question outdated assumptions, use best-of-breed technology

DEVELOPMENT

- Six compute-only nodes
- Two nodes for Gluster storage
- Apache Spark running under Apache Mesos
- Open-source “notebook” interfaces to analyses



DATA SOURCES

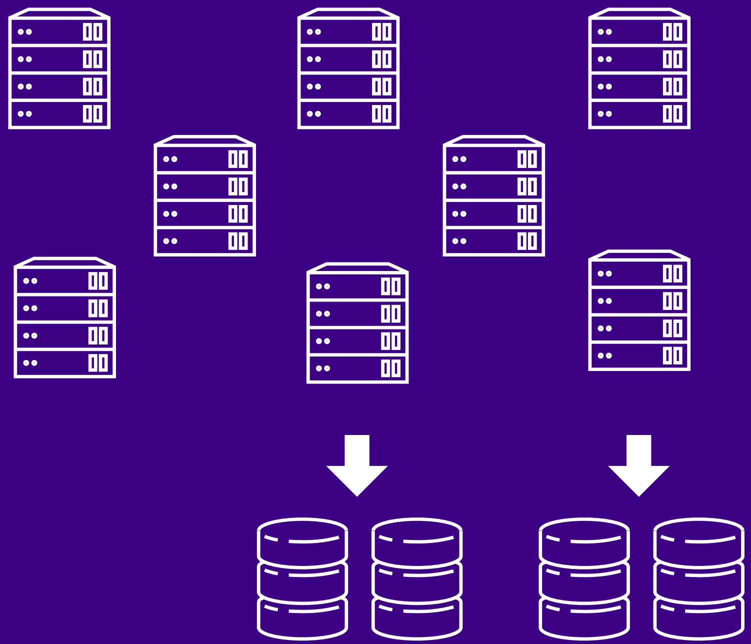
FTP

S3

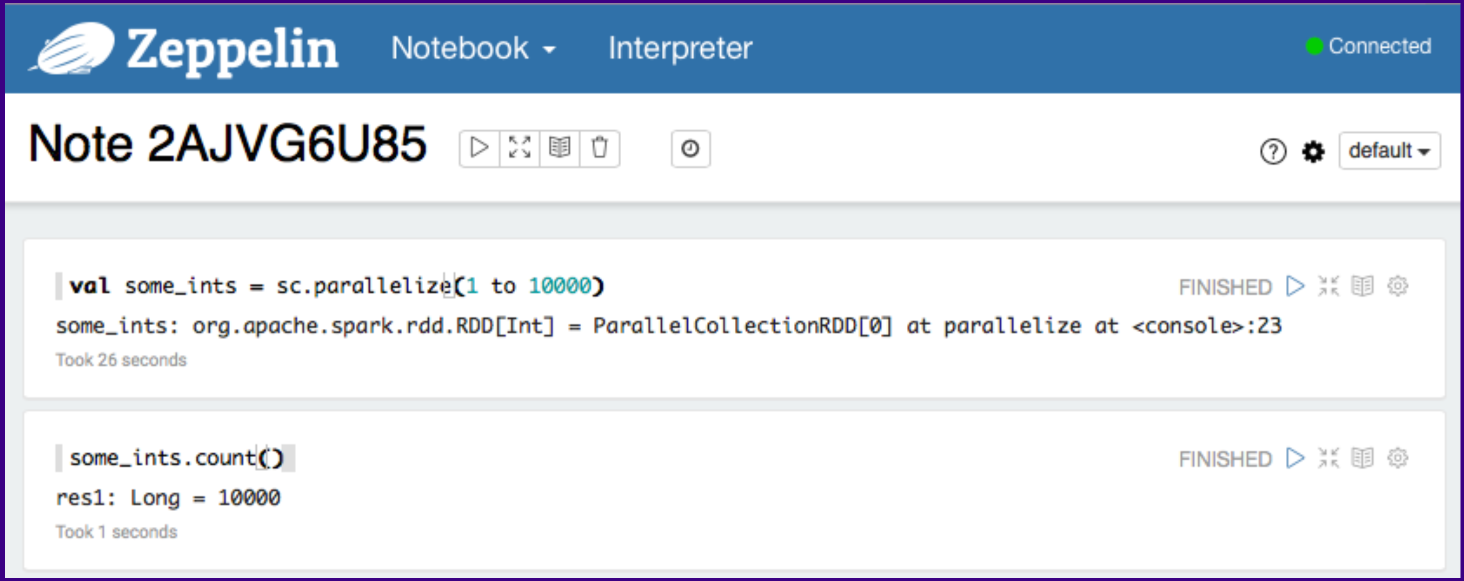
SQL

MongoDB

ElasticSearch



INTERACTIVE QUERY



TWO CASE STUDIES

ROLE ANALYSIS

- Data source: historical configuration and telemetry data for internal machines from ElasticSearch
- Data size: hundreds of GB
- Analysis: identify machine roles based on the packages each has installed

BUDGET FORECASTING

- Data sources: operational log data for OpenShift Online (from MariaDB), actual costs incurred by OpenShift
- Data size: over 120 GB
- Analysis: identify operational metrics most strongly correlated with operating expenses; model daily operating expense as a function of these metrics

Aggregating performance metrics:
17 hours in MariaDB, 15 minutes in Spark!

NEXT STEPS

DEMO VIDEO

See a video demo of Continuum Analytics, PySpark, and Red Hat Storage: h.264 or Ogg Theora

WHERE FROM HERE

- Check out the Emerging Technology Data Science team's library to help build your own data-driven applications: https://github.com/willb/silex/
- See my blog for articles about open source data science: http://chapeau.freevariable.com
- Questions?


THANKS

redhat®