ISTM 6212 - Week 12
Hands on w/Spark

Daniel Chudnov, dchud@gwu.edu

2016-11-29

Agenda

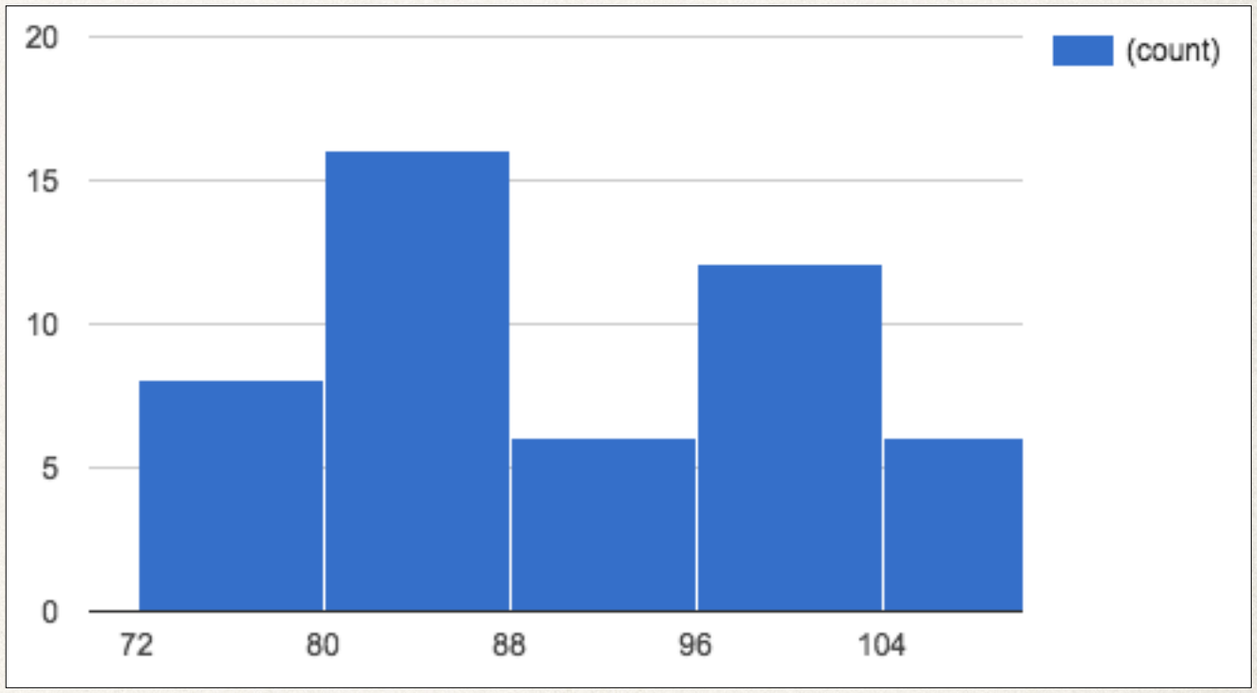
- Schedule check
- Project 02 / Trifacta feature / Project 03 check-in
- Spark - RDDs, DataFrames, SQL
- Streams and Column storage
- Spark w/Zeppelin

Schedule check

Project 02 wrap-up

Project 02 - Overall

- Bimodal performance
- $\mu = 89.8$
- A few common mistakes



Project 02 - More good examples

- Yuanjing and Youdan's exploration:
  - github.com/Yuanjing-Han/Yuanjing-Han-istm-6212/blob/master/project-02%2BYuanjing_and_Youdan.ipynb
- Daniel and Kevin's encoding lesson

Project 02 - Common mistakes

- Describe your data! What is it, where's it from, what does it cover? Include a link to a main site.

- Create dimensions with DISTINCT values

- Leave precise time (minutes, seconds) out of dimensions

- Use JOINs and UPDATEs to populate surrogate keys in fact tables

- At most one SERIAL per table

Project 02 - Common mistakes

- %matplotlib inline # <— only need this once
- If time is on the X axis, order it by time
- Assume your reader is not at your computer
- Keep and share a copy of your dataset

Project 02 - Common mistakes

- Use your spellchecker
- Use correct punctuation
- Never, ever write slang (e.g. “gonna”)
- “I” vs. “We”

Project 02 - Data woes

- Ars longa, data brevis*

- *Ask Abhinav, Aida, Boer, Clare, Cora, Gaoshuang, Junfei, Livia, Xinyi L, Xinyi W

Project 02 - Discussion

- What about street addresses?
  - Natural affinity?
  - Filtering function?

Trifacta feature

Project 03 check-in

Spark RDDs, Dataframes, SQL

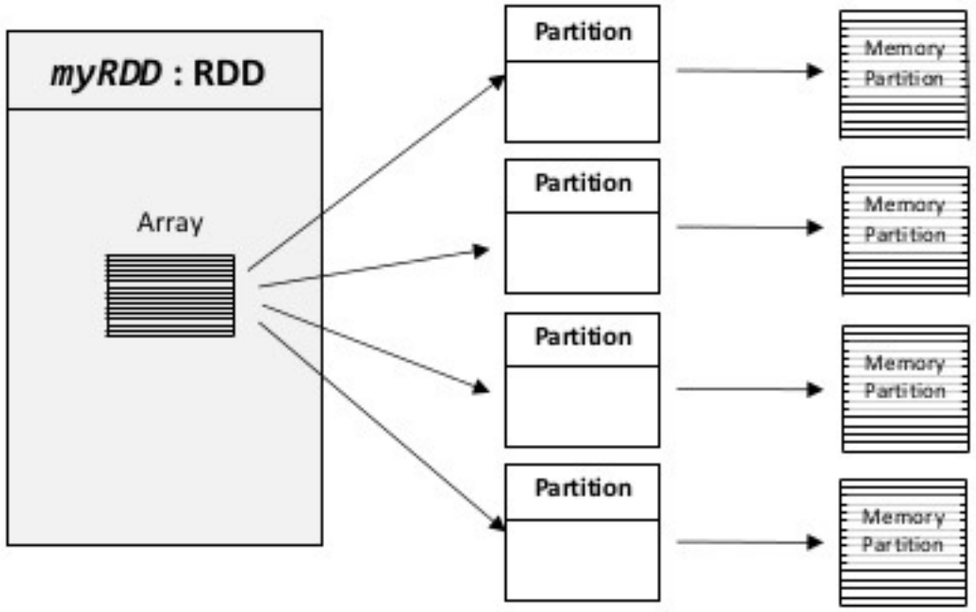
Running Spark yourself

- My recommendation: Install it yourself, directly on your machine - **not** in a VM, and use PySpark
- Or use Databricks Community Edition
- Or run it on AWS EMR or EC2 w/Zeppelin
- Or ask Lawrence about running it w/Docker

Spark RDDs

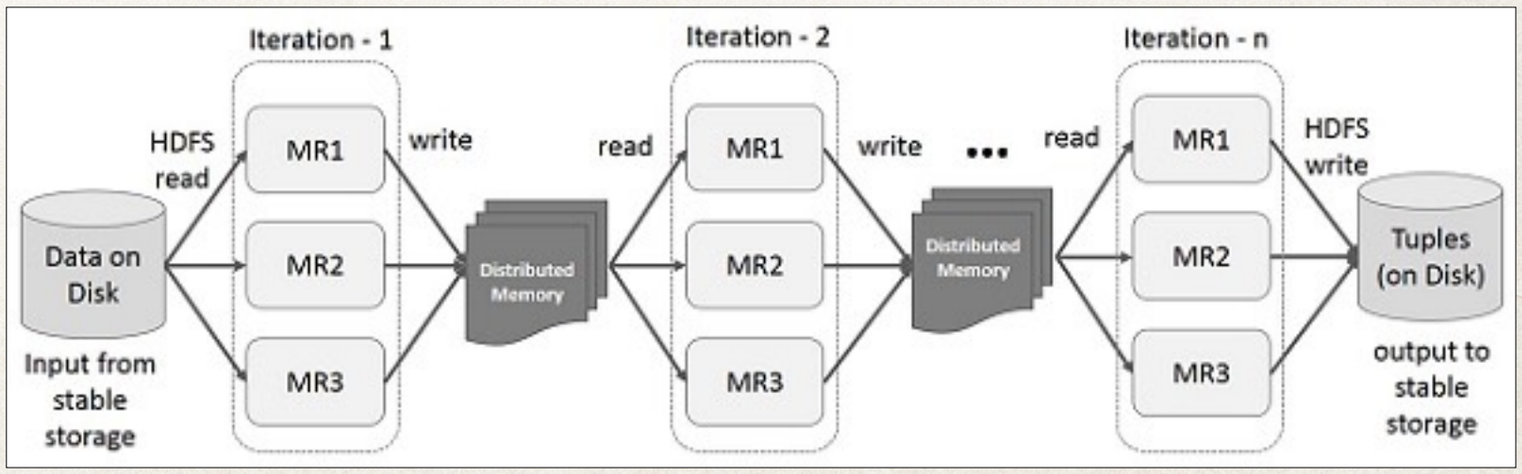
www.slideshare.net/sparkInstructor/apache-spark-rdd-101

What is an RDD?



Some RDD Characteristics

- Hold references to Partition objects
- Each Partition object references a subset of your data
- Partitions are assigned to nodes on your cluster
- Each partition/split will be in RAM (by default)



www.tutorialspoint.com/apache_spark/apache_spark_rdd.htm

Spark RDDs

```
many_texts = sc.textFile('texts/*.txt')
word_counts = many_texts \
    .flatMap(lambda line: line.lower().split(" ")) \
    .filter(lambda word: word not in stop_words) \
    .map(lambda word: (word, 1)) \
    .reduceByKey(lambda a, b: a + b)
word_counts.takeOrdered(25, lambda w: -w[1])
```

RDD —> DataFrames

- RDDs are a key innovation from Spark
- R- and Pandas-like DataFrames suit many tasks
- SQL suits many tasks
- w/Spark 2.0+, DataFrames are key concept

Spark DataFrames

- Like R / Pandas, mostly
- Fully supported in pyspark
- Some RDD-like residuals

```
# Select everybody, but increment the age by 1
df.select(df['name'], df['age'] + 1).show()
# +-------+---------+
# |   name|(age + 1)|
# +-------+---------+
# |Michael|     null|
# |   Andy|       31|
# | Justin|       20|
# +-------+---------+

# Select people older than 21
df.filter(df['age'] > 21).show()
# +---+----+
# |age|name|
# +---+----+
# | 30|Andy|
# +---+----+

# Count people by age
df.groupBy("age").count().show()
# +----+-----+
# | age|count|
# +----+-----+
# |  19|    1|
# |null|    1|
# |  30|    1|
# +----+-----+
```

spark.apache.org/docs/latest/sql-programming-guide.html

Spark SQL

- DataFrame select/group/filter implementation shares a lot of concerns with RDBMS/SQL implementation:
  - Query parsing, planning, optimization
- When those are in place for one, you can handle the other
- SQL works in Spark wherever DataFrames work

DataFrame —> SQL

```
# Register the DataFrame as a SQL temporary view
df.createOrReplaceTempView("people")

sqlDF = spark.sql("SELECT * FROM people")
sqlDF.show()
# +----+-------+
# | age|   name|
# +----+-------+
# |null|Michael|
# |  30|   Andy|
# |  19| Justin|
# +----+-------+
```

- Same DataFrame (df)
- df.createOrReplaceTempView(“TABLE_NAME”)

Streams and Column storage

Streaming data

- "A stream is just more table data you don't have yet."
  -paraphrasing Jennifer Widom cs.stanford.edu/people/widom/
- Repeating similar process in batches
- Key question: what to do next with the data?

Streaming example: Tweets

```
sc = SparkContext()
ssc = StreamingContext(sc,
        app.config['SPARK_BATCH_INTERVAL'])

# Watch for new files in args.dir
incoming_tweets = ssc.textFileStream(args.dir)

# Send each new batch through process()
incoming_tweets.foreachRDD(process_tweets)

# Kick off directory watching
ssc.start()
```

Streaming example: Tweets (cont'd)

```
def process_tweets(tweet_rdd):
    tweet_json = tweet_rdd.map(lambda line:
                               json.loads(line)).cache()
    hashtag_counts = tweet_json.flatMap(
        lambda tweet: [(ht['text'].lower(), 1)
            for ht in tweet.get(‘entities',
                {'hashtags': []})['hashtags']]) \
    .reduceByKey(lambda a, b: a + b).collect()
pipe = redis_conn.pipeline()
for hashtag, count in hashtag_counts:
    pipe.zincrby(timed_name, hashtag, count)
pipe.execute()
```

Row storage and Column storage

- In general, RDBMS systems store data in rows
- Think of CSV:
  - Record 1: field_1, field_2, field_3
  - Record 2: field_1, field_2, field_3
- Datatype definitions (DDL) define row / attr sizes

Column storage

- For many analytics applications, we are measuring facts given one or more context:
  - Filter and group by region, district, brand, and product, then add up sales
  - Filter and group by department, professor, and year, than count enrollment by semester
- Those facts we measure are often “one column”

Table

	Country	Product	Sales
Row 1	India	Chocolate	1000
Row 2	India	Ice-cream	2000
Row 3	Germany	Chocolate	4000
Row 4	US	Noodle	500

Row Store

Row 1	India
	Chocolate
	1000
Row 2	India
	Ice-cream
	2000
Row 3	Germany
	Chocolate
	4000
Row 4	US
	Noodle
	500

Column Store

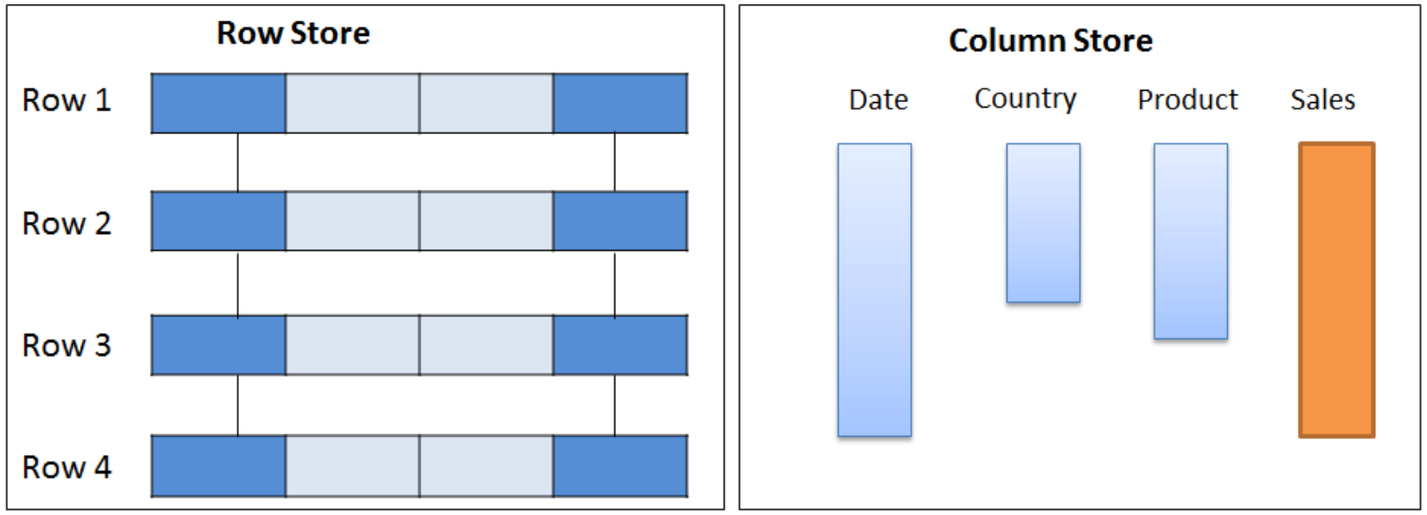
Country	India
	India
	Germany
	US
Product	Chocolate
	Ice-cream
	Chocolate
	Noodle
Sales	1000
	2000
	4000
	500

saphanatutorial.com/column-data-storage-and-row-data-storage-sap-hana/

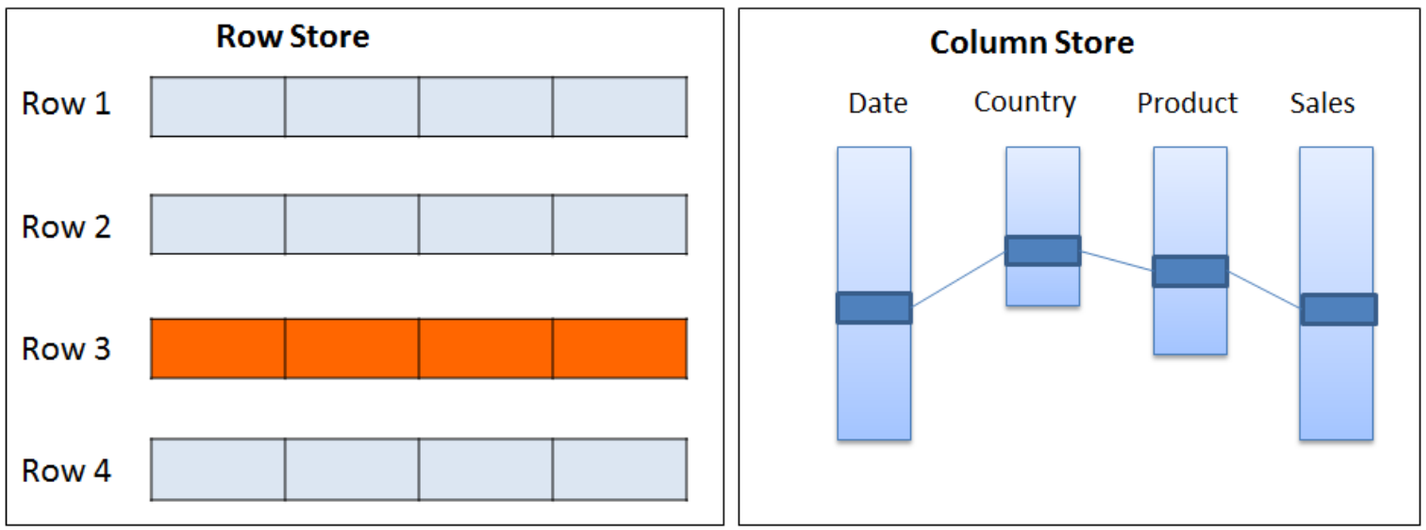
Table - SALES

	Date	Country	Product	Sales
Row 1	2013-01-01	India	Chocolate	1000
Row 2	2013-01-10	India	Ice-cream	2000
Row 3	2013-02-20	Germany	Chocolate	4000
Row 4	2013-03-01	US	Noodle	500

Column Operation: SELECT SUM(SALES) FROM SALES WHERE DATE > 2012-01-01



Row Operation: SELECT * FROM SALES WHERE COUNTRY = 'INDIA'



saphanatutorial.com/column-data-storage-and-row-data-storage-sap-hana/

Benefits of column storage

- Very fast measurements / aggregations
- Column indexing: each block start/finish values allow skipping to precise blocks
- Compression, indexing advantages for speed

Column storage in practices

- Many data warehouse products use column storage
- AWS Redshift is a version of PostgreSQL optimized w/column storage
- Parquet is a columnar file format gaining popularity w/Hadoop, Spark, etc. computing

Spark w/Zeppelin

tinyurl . com / dchud - spark - demo