Center for Data Science

Week 05.2: Using Spark

DS-GA 1004: Big Data

Apache Spark (2009, v0 2012)

- Cluster computing framework using RDDs
- Integrates with Hadoop ecosystem
  - HDFS for storage (but other backends are possible)
  - Mesos or (Hadoop) YARN for scheduling
- Written in Scala with API in other languages
  - Python, Java, R, etc

Architecture: session and driver

- **Driver** is the process that you run on, e.g., the head / login node

- The **Session** object connects **your code** to the cluster / compute nodes

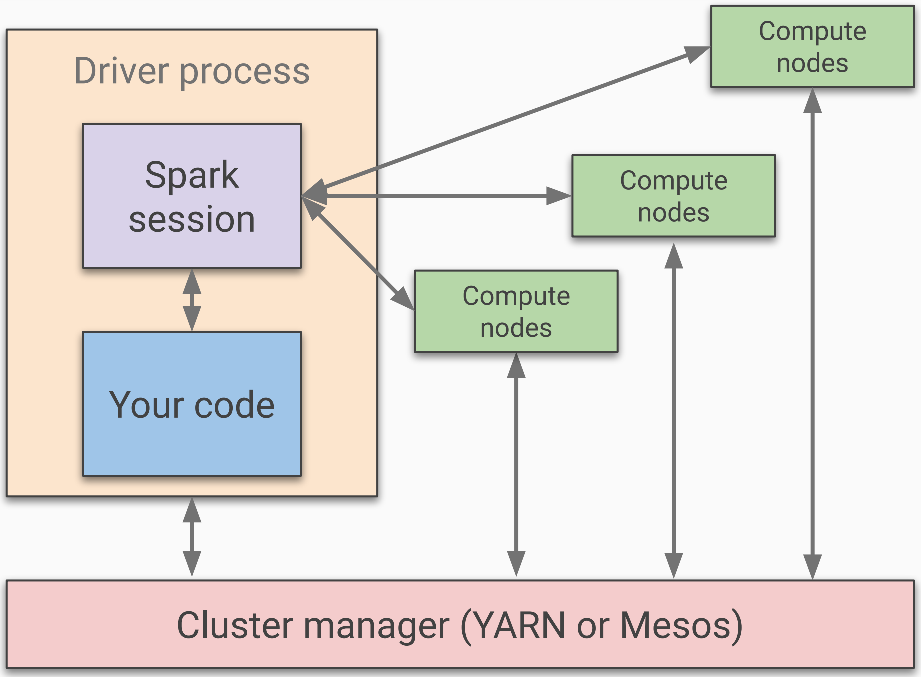


Figure adapted from [Chambers and Zaharia, 2018]

Aside: Why Scala?

- RDD design fits well with functional programming
  - Closures (function + environment) encapsulate everything you need to construct a result
  - Lazy evaluation
  - Immutable data
- Scala compiles to Java virtual machine (JVM)
  - JVM byte code is portable across machines
  - Integration with Hadoop tools (in Java) is relatively easy

Aside part 2: closures

```
function make_closure(x):
    function f(y):
        return x + ' stomps on ' + y
    return f

stomp ← make_closure('Claosaurus')

print(stomp('the village'))
```

- **Closures** are a functional programming construction that combine a function with its environment (ie dependencies)
- Does this sound like **RDDs**?
- Example code (⇐) constructs and then executes a closure (**stomp**)
- Scala's a great language for this!

https://prehistorico.fandom.com/es/wiki/Claosaurus

Example: gradient descent revisited

```
val points = spark.textFile(...).map(parsePoint).cache()
var w = Vector.random(D)

for (i ← 1 to ITERATIONS)
    val grad = spark.accumulator(new Vector(D))

    for (p ← points)
        val grad_p = ∇_w f(p ; w)
        grad += grad_p

    w -= grad.value
```

- Some scala notation:
  - **val** means immutable **val**ue
  - **var** means mutable **var**iable
- Outer loop runs in series
- Inner loop runs in parallel over points
  - Equivalent to:
    points.**foreach**(p ⇒ {loop body})
- **grad** is a shared *accumulator*
  - Write-only data structure for associative/commutative updates

Beyond Scala

- You don't need to code Scala to use Spark
- Spark can run from R or Python (or Java)
- Beware: R and Python may not be as fast as Scala
- **Crossing process boundaries** can be expensive, but Spark does a good job of managing this

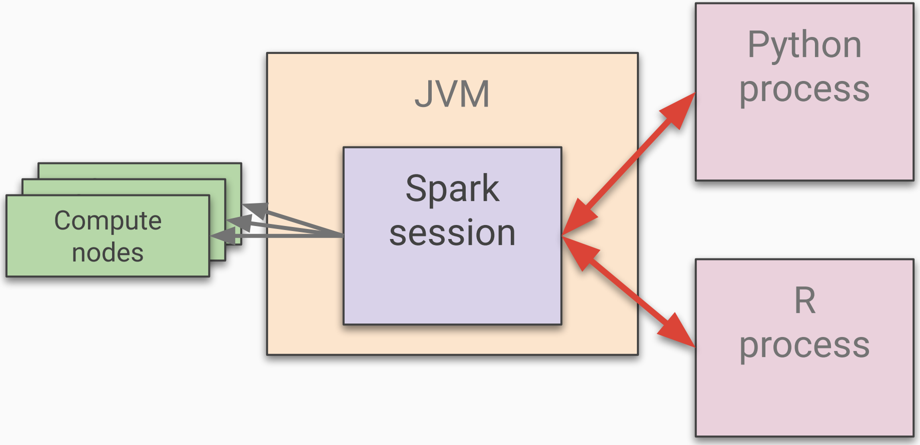
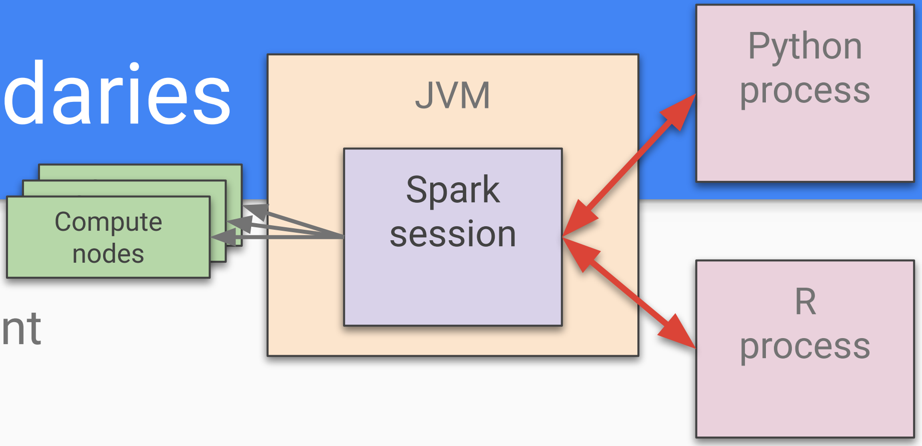


Figure adapted from [Chambers and Zaharia, 2018]

Crossing process boundaries

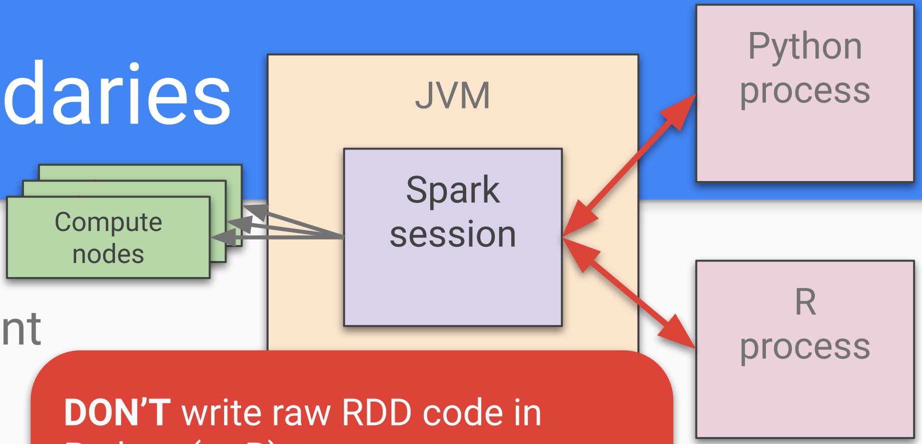


- Imagine rewriting the gradient descent loop in Python
- Technically it's possible, but it's **slow**
- This is because each operation needs to jump out of Scala and into Python, serializing all data **between processes**

```
val points =
spark.textFile(...).map(parsePoint).cache()
var w = Vector.random(D)

for (i ← 1 to ITERATIONS)
        val grad = spark.accumulator(new Vector(D))
        for (p ← points)
                val grad_p = ∇w f(p ; w)
                grad += grad_p
        w -= grad.value
```

Crossing process boundaries



- Imagine rewriting the gradient descent loop in Python
- Technically it's possible, but it's **slow**
- This is because each operation needs to jump out of Scala and into Python, serializing all data **between processes**

DON'T write raw RDD code in Python (or R)

DO use existing packages written in Scala with Python bindings

```
ache()

for (i ← 1 to ITERATIONS)
    val grad = spark.accumulator(new Vector(D))
    for (p ← points)
        val grad_p = ∇w f(p ; w)
        grad += grad_p
    w -= grad.value
```

Spark DataFrames API

- RDDs are great, but a bit cumbersome for ad-hoc computation
- **DataFrames** are common representations in many languages
  - R, pandas (Python), etc.
- Spark 2.x added a DataFrame API as a primary interface
  - Code looks more or less like pandas/Python!

DataFrames and RDDs

- DataFrames in Spark are like relations in RDBMS
  - Well-defined schema with types over columns
  - Each row is a tuple (sort of...)

- DataFrames operations are translated into RDD transformations by Spark

- RDD transformations can then be executed within JVM
  - No more serialization of data between JVM⇔Python!

Spark-SQL

- Spark 2.x allows you to express queries in SQL
  - Or using an object-method chaining API — the two are equivalent!
- Queries are executed against DataFrames
  - DataFrames are secretly RDDs, not RDBMS tables!
- Queries can be optimized by analyzing the RDD lineage graph

```
df.createOrReplaceTempView('my_table')

res = spark.sql('  SELECT zip_code, sum(height) as H
                   FROM my_table
                   GROUP BY zip_code')
res.show()
```

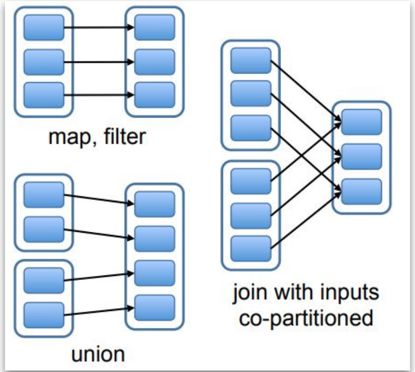
```
df.groupBy('zip_code')
  .sum('height')
  .withColumnRenamed('sum(height)', 'H')
  .show()
```

Repartitioning

- Sometimes you know in advance which columns of a DataFrame will be filtered
    - E.g., dates or timestamps

- You can give hints to Spark that RDD partitions should align accordingly
    - df.repartition(# PARTITIONS, col(“NAME OF COLUMN”))
    - This can reduce the width of RDD dependencies

- **This is much like indexing in RDBMS**



Tips and pitfalls

- Before running an action, run the **explain**() method on the DataFrame
  - This will give you an execution plan
  - You might identify some inefficiencies or bugs this way

- Be careful with **collect**()!
  - This will stream all results back to the driver node
  - If it's a large data set, and you forgot an aggregation step, this will be very bad news.
  - Test-drive a large query with **take(10)** instead of **collect**()
  - Probably you want **.save()** instead of **.collect()** anyway

Wrap-up on Spark

- RDD framework is more flexible than Map-Reduce
- Caching can make interactive jobs faster
- SparkSQL / DataFrames API makes development easy