Jakub Háva
jakub@h2o.ai

Introduction to Spark

The Amsterdam Pipeline Factory of Data Science, Amsterdam
December 6, 2016

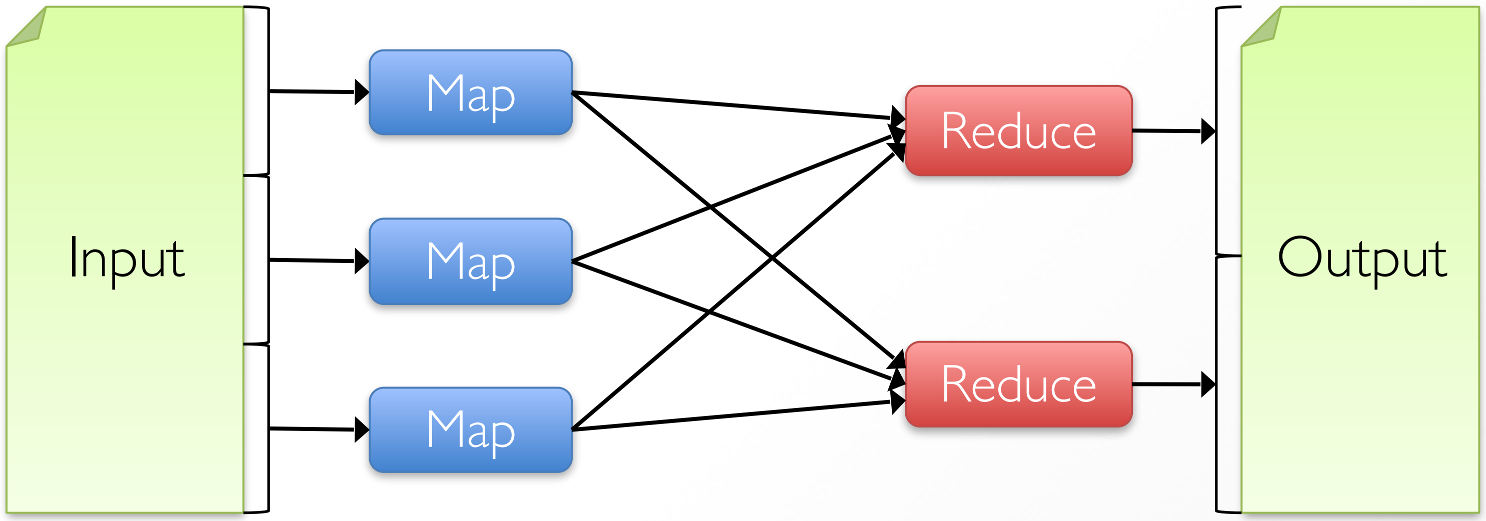
Who am I

- Finishing high-performance cluster monitoring tool for JVM based languages (JNI, JVMTI, instrumentation)
- Finishing Master’s at Charles Uni in Prague
- Software engineer at H2O.ai - Core Sparkling Water

- Tea lover (doesn’t mean I don’t like beer!)

Data Flow in typical MapReduce

- Acyclic data flow, from storage A to storage B
- Benefit is that runtime can decide where to run tasks and can automatically recover in case of their failures



Inefficiencies of acyclic Data Flow

- Inefficient for application that repeatedly reuse a *working set* of data
  - Iterative algorithms - machine learning
  - Interactive data mining tools (R, Scala interpreter)
- With typical MapReduce, the data are reloaded for each query

Solution: RDD

RDD: Resilient Distributed Dataset

- Allow apps to keep working sets in memory for efficient reuse
- Whilst keep the benefits of MapReduce
  - Fault tolerance, data locality, scalability
- General approach

Spark Programming Model

Spark Runs on

- YARN, Mesos, Standalone cluster
- 1 driver node
- Several executor nodes
  - can be started dynamically when needed based on provided resources by resource manager

RDDs

- Immutable, partitioned collections of objects
- Created through parallel transformations
  - **map**, **filter**, **groupBy**, **join**…
  - Can be cached for efficient use on various levels
- Transformations does not start any computations - lazy behaviour
- Actions start the computations
  - **count**, **reduce**, **collect**, **save**
- RDDs are split across multiple partitions on multiple nodes

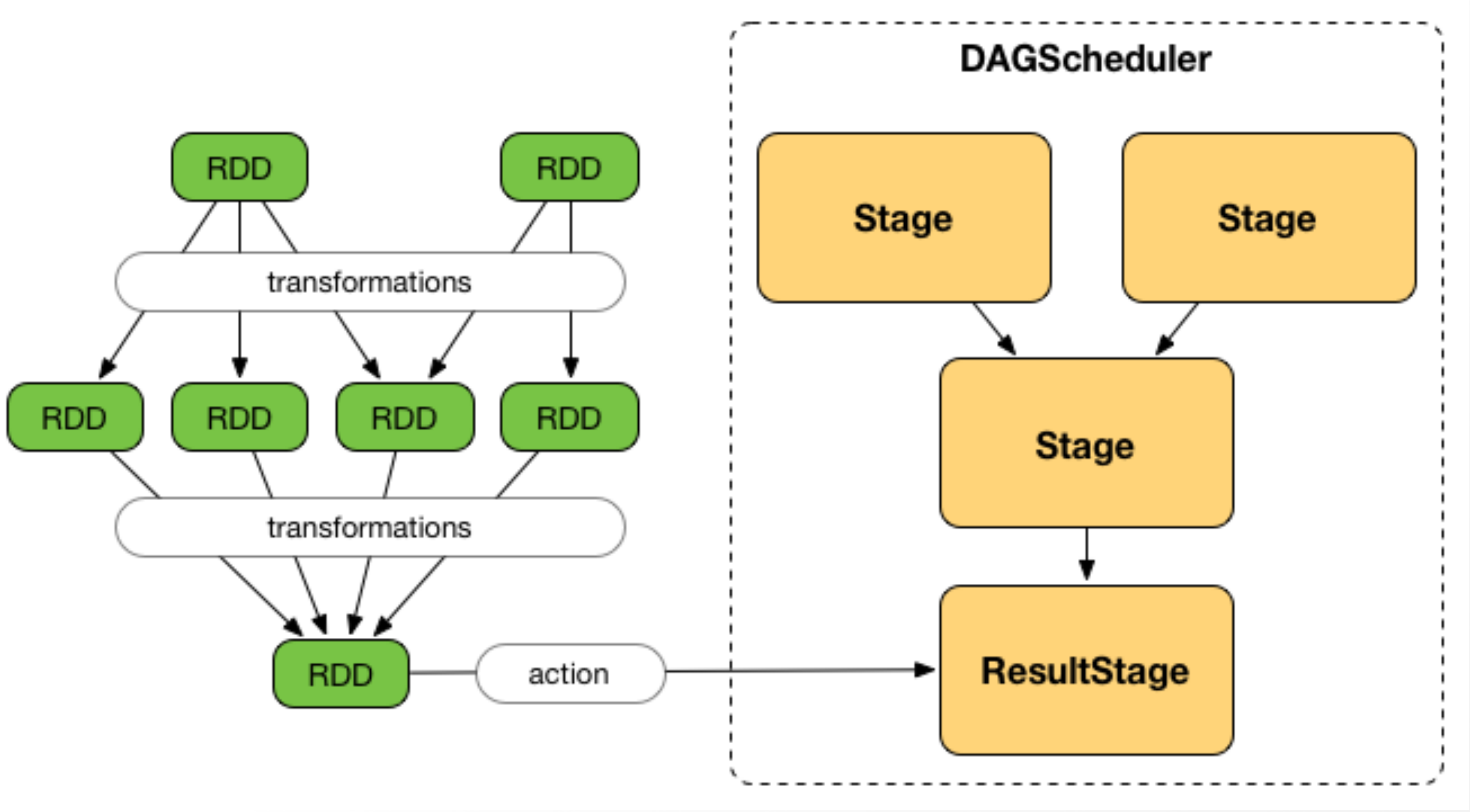
RDD lineage

- Result of applying transformations on the RDD
- val rdd = sc.textFile(...)
- val filtered = rdd.filter(...)
- **toDebugString** method on RDD - see the lineage
- Plan describing the computation

Spark Job

- Starts with an action called on RDD
- Job is split into multiple stages (depends on type of transformations - **stage boundaries**)
- Stage is executed by DAG Scheduler
- Each stage is split into several tasks (usually #partitions in the stage)

DAG Scheduler



Abstractions on top of RDD

RDD

- Low-level transformations
- Data is unstructured
- Good when we want to manipulate with data using functional programming constructs
- Don't have schema of the data
- On heap, thus affected by GC and are using Java Serialisation

RDD Example

```
rdd.filter(_.age > 21)               // transformation
   .map(_.last)                      // transformation
   .saveAsObjectFile("under21.bin") // action
```

```
rdd.filter(_.age > 21)
```

Dataframes

- Build on top of RDDs => immutable distributed collection of data
- Data organised into columns like a table in relational database
- Have a schema
- DSL and SQL like specific language
- Better performance
  - Can use the schema and a lot of optimilizations - Spark Query Optimiser
- Can be stored off-heap, Spark specific serialisation

Datarame example

```
df.filter("age > 21")
```

```
df.filter(df.col("age").gt(21))
```

Dataset

- First appeared in Preview of Spark 1.6
- Attempt to bring the best from RDDs and DataFrames
  - Compile type safety as in RDD
  - Performance as on DataFrames
- Off-heap storage (outside the JVM memory)
- Usage of encoders to translate between JVM representation and Spark internal binary format, no need to de-serialise the object in order to read it

Dataset example

```scala
val sc = new SparkContext(conf)
val sqlContext = new SQLContext(sc)
import sqlContext.implicits._
val sampleData: Seq[ScalaPerson] = ScalaData.sampleData()
val dataset = sqlContext.createDataset(sampleData)
```

```scala
dataset.filter(_.age < 21);
```

Thank you!

Sparkling Water is
open-source
ML application platform
combining
power of Spark and H2O

Learn more at h2o.ai
Follow us at @h2oai