Class 4

New York University

Summer 2017

1. Review
2. Project Discussion
3. Project Breakout
4. Spark RDD Design
5. Scala Tuples in Spark

1. Review
2. Project Discussion *(see BDAD_SU17_ProjectInfo.pdf)*
3. Project Breakout
4. Spark RDD Design
5. Scala Tuples in Spark

1. Review
2. Project Discussion
3. Project Breakout
4. Spark RDD Design
5. Scala Tuples in Spark

1. Review
2. Project Discussion
3. Project Breakout
4. Spark RDD Design
5. Scala Tuples in Spark

- Spark processing involves one of the following actions
    - Creating new RDDs
    - Transforming existing RDDs
    - Calling operations on RDDs to compute a result

- Under the hood, Spark distributes the data contained in RDDs
    - Data is distributed automatically to cluster servers
    - The operations you specify are parallelized across the cluster too

- An RDD is an immutable, distributed collection of objects
- An RDD contains
  - Python, Scala, or Java objects
  - It can even contain objects from user-defined classes

- Each RDD object is referred to as a *partition*
    - Specified computations are performed on each partition
    - Partitions can exist on any of the worker nodes in the cluster

- RDDs are created by
    - Loading an external dataset
    - Distributing a collection across the cluster
    - Transforming an existing RDD into a new RDD

- As discussed in an earlier class, RDDs offer two types of operations
  - *Transformations* construct a new RDD from an existing one
  - *Actions* compute a result based on processing an RDD
    - Actions do not create a new RDD
    - Actions return a result to the driver program or write the result to storage

- Transformations are performed lazily
  - I.e., when an action is issued, this is when all transformations are executed
  - This allows Spark to optimize operations to reduce the amount of data processed whenever possible
  - For example, **`first()`** causes Spark to read the first line only, not the entire (potentially Big Data) file
    - Contrast with MapReduce, where the entire file would be read

- Spark recomputes the related RDDs each time you run an action on them
    - You can avoid recomputing by using **`RDD.persist()`** (or **`cache()`**)
- After the first computation has been performed, the RDD can be persisted to memory (for example)
    - The RDD is persisted as partitions across the cluster
    - Once persisted, the data is available for future operations without having to be recomputed
- RDDs can also be persisted to disk

- Data is not persisted by default
    - Spark allows the programmer to control what should be persisted, and what need not be persisted
    - If data were persisted by default, there would be times when storage and time would be wasted to store data even though it would only be used once!
- We will look at options for persisting in a future class

- What is the number of partitions created?
  - By default, the number of partitions created for an RDD created from an HDFS file is the number of blocks in the file
  - If you want to verify the number of partitions created, use the **`partitions`** method of RDDs

- To check the number of partitions:

```
scala> someRDD.partitions.size
res0: Int = 30
```

- Here is an example that creates 30 partitions:

```
someRDD = sc.parallelize(range(101),30)
```

1. Review
2. Project Discussion
3. Project Breakout
4. Spark RDD Design
5. Scala Tuples in Spark

A Tuple is

- A group of individual values that can be treated as a single entity
- A compile-time entity

Common uses of Tuples

- For returning more than one value from a function
- Key-value pairs
- For sending multiple values in a single message between concurrent processes
- For buffering a data record / related data of varying types

Limitations of Tuples

- The number of values in a Tuple cannot be changed after it is initialized

- Tuples consist of between two (min) and twenty-two (max) values

A tuple containing two fields is a Tuple2, or *pair*

- Explicit declaration of a Tuple2 variable

```
val myTup2A = Tuple2(4, "iFruit")
> myTup2A: (Int, String) = (4,iFruit)

myTup2A.getClass
> Class[_ <: (Int, String)] = class scala.Tuple2
```

- The -> syntax is available for Tuple2, but not for larger tuples

```
val myTup2B = 4 -> "iFruit"
> myTup2B: (Int, String) = (4,iFruit)
```

- You can also allow the type to be inferred

```
val myTup2C = (4, "iFruit")
> myTup2C: (Int, String) = (4,iFruit)
```

- Tuples contain two or more fields
- Individual tuple values can be accessed with _1, _2 syntax
- Notice that tuples are ***one***-based

```
val myTup2B =  4 -> "iFruit"
> myTup2B: (Int, String) = (4,iFruit)

myTup2B._1
> Int = 4

myTup2B._2
> String = iFruit
```

- *swap* is syntactic sugar that works for Tuple2 only

```
myTup2B.swap
> (String, Int) = (iFruit,4)
```

- Tuples with more than 2 elements: **`Tuple`*`N`***
- Declaring **`TupleN`** variables is similar to declaring **`tuple2`** variables
- Use **`_n`** to access values in a **`TupleN`**

```
val myTup = (4,"MeToo","1.0",37.5,41.3,"Enabled")
> myTup: (Int, String, String, Double, Double, String) =
   (4,MeToo,1.0,37.5,41.3,Enabled)

myTup.getClass
> Class[_ <: (Int, String, String, Double, Double, String)] =
  class scala.Tuple6

println( myTup._3 + " / " + myTup._5 )
> 1.0 / 41.3
```

- Use **`productPrefix`** to get the tuple's class name as a string
- Use **`productArity`** to get the tuple size as an integer

```
val oneRecord = ("2014-03-15:10:10:20", "MeeToo", 3.0,
"8316b507-7620-47aa-b56b-cae5cb2cd819", 0, 19, 69, 31, 51, 44, "TRUE",
"enabled" ,"disabled", 33.4467594, -111.3653269)
> oneRecord: (String, String, Double, String, Int, Int, Int, Int, Int,
Int, String, String, String, Double, Double) =
(2014-03-15:10:10:20,MeeToo,3.0,8316b507-7620-47aa-b56b-cae5cb2cd819,
0,19,69,31,51,44,TRUE,enabled,disabled,33.4467594,-111.3653269)

oneRecord.productPrefix
> String = Tuple15

oneRecord.productArity
> Int = 15
```

- The values in a tuple can be converted to a single string using **toString**
  - Note that parentheses and commas are part of the new string

```
oneRecord._4
> String = 8316b507-7620-47aa-b56b-cae5cb2cd819

oneRecord.toString
> String = (2014-03-15:10:10:20,MeeToo,3.0,8316b507-7620-47aa-b56b-
cae5cb2cd819,
0,19,69,31,51,44,TRUE,enabled,disabled,33.4467594,-111.3653269)
```

- Convert a string to a tuple using **partition**
- The **partition** function
  - Accepts a condition
    - In the example below, the condition is 'isUpper'
  - Returns a Tuple2 where the first value is what satisfied the condition
    - The second value contains what did not satisfy the condition

```
val myTup = ("Oranges", "Bananas", "apples", "Guavas")

val myStr = myTup.toString
> myStr: String =
 (Oranges,Bananas,apples,Guavas)

val myTup2 = myStr.partition(_.isUpper)
> (String, String) = (OBG, (ranges,ananas,apples,uavas))

val sortedValues = myTup2._1.sorted
> String = BGO
```

Homework

See homework packet.