

Big data for internet applications

RDD-based programming

Transformations on two RDDs of key-value pairs

Transformations on two RDDs of pairs

- Spark supports also some transformations that are applied on two RDDs of key-value pairs at the same time
 - `subtractByKey`, `join`, `coGroup`, etc.

SubtractByKey transformation

SubtractByKey transformation

- Goal
 - Create a new RDD of key-value pairs containing only the pairs of the first input RDD of pairs associated with a key that is not appearing as key in the pairs of the second input RDD or pairs
 - The data type of the new RDD of pairs is the same of the “first input” RDD of pairs
 - The two input RDD of pairs must have the same type of keys
 - The data type of the values can be different

SubtractByKey transformation

- Method
 - The subtractByKey transformation is based on the `subtractByKey(other)` method of the `RDD` class
 - The two input RDDs of pairs analyzed by `subtractByKey` are the one on which the method is invoked and the one passed as parameter (i.e., `other`)

SubtractByKey transformation: Example

- Create two RDDs of key-value pairs from two local python lists
 - First list – Profiles of the users of a blog (username, age)
 - [("PaoloG", 40), ("Giorgio", 22), ("PaoloB", 35)]
 - Second list – Banned users (username, motivation)
 - [("PaoloB", "spam"), ("Giorgio", "Vandalism")]
- Create a new RDD of pairs containing only the profiles of the non-banned users

SubtractByKey transformation: Example

```
# Create the first local python list
profiles = [ ("PaoloG", 40), ("Giorgio", 22), ("PaoloB", 35)]

# Create the RDD of pairs from the profiles local list
profilesPairRDD = sc.parallelize (profiles)

# Create the second local python list
banned = [ ("PaoloB", "spam"), ("Giorgio", "Vandalism")]

# Create the RDD of pairs from the banned local list
bannedPairRDD = sc.parallelize (banned)

# Select the profiles of the "good" users
selectedProfiles = profilesPairRDD.subtractByKey(bannedPairRDD)
```

Join transformation

Join transformation

- Goal
 - Join the key-value pairs of two RDDs of key-value pairs based on the value of the key of the pairs
 - Each pair of the input RDD of pairs is combined with all the pairs of the other RDD of pairs with the same key
 - The new RDD of key-value pairs
 - Has the same key data type of the “input” RDDs of pairs
 - Has a tuple as value (the pair of values of the two joined input pairs)
 - The two input RDDs of key-value pairs
 - Must have the same type of keys
 - But the data types of the values can be different

Join transformation

- Method
 - The join transformation is based on the `join(other)` method of the `RDD` class
 - The two input RDDs of pairs analyzed by join are the one on which the method is invoked and the one passed as parameter (i.e., `other`)

Join transformation: Example

- Create two RDDs of key-value pairs from two local python lists
 - First list – List of questions (QuestionId, Text of the question)
 - [(1, "What is .. ?"), (2, "Who is ..?")]
 - Second list – List of answers (QuestionId, Text of the answer)
 - [(1, "It is a car"), (1, "It is a byke"), (2, "She is Jenny")]
- Create a new RDD of pairs to associate each question with its answers
 - One pair for each possible pair question - answer

Join transformation: Example

```
# Create the first local Python list
questions= [(1, "What is .. ?"), (2, "Who is ..?")]

# Create the RDD of pairs from the local list
questionsPairRDD = sc.parallelize(questions)

# Create the second local python list
answers = [(1, "It is a car"), (1, "It is a byke"), (2, "She is Jenny")]

# Create the RDD of pairs from the local list
answersPairRDD = sc.parallelize(answers)

# Join questions with answers
joinPairRDD = questionsPairRDD.join(answersPairRDD)
```

Join transformation: Example

```
# Create the first local Python list  
questions= [(1, "What is .. ?"), (2, "Who is ..?")]
```

```
# Create the RDD of pairs from the local list  
questionsPairRDD = sc.parallelize (questions)
```

The key part of the returned RDD of pairs is an integer number

The value part of the returned RDD of pairs is tuple containing two values: (question, answer)

```
# Join questions with answers  
joinPairRDD = questionsPairRDD.join(answersPairRDD)
```

CoGroup transformation

Cogroup transformation

- Goal
 - Associated each key **k** of the two input RDDs of key-value pairs with
 - The list of values associated with **k** in the first input RDD of pairs
 - And the list of values associated with **k** in the second input RDD of pairs
 - The new RDD of key-value pairs
 - Has the same key data type of the two “input” RDDs of pairs
 - Has a tuple as value (the two lists of values of the two input RDDs of pairs)
 - The two input RDDs of key-value pairs
 - Must have the same type of keys
 - But the data types of the values can be different

Cogroup transformation

- Method
 - The cogroup transformation is based on the `cogroup(other)` method of the `RDD` class
 - The two input RDDs of pairs analyzed by cogroup are the one on which the method is invoked and the one passed as parameter (i.e., `other`)

Cogroup transformation: Example

- Create two RDDs of key-value pairs from two local python lists
 - First list – List of liked movies (userId, likedMovies)
 - [(1, "Star Trek"), (1, "Forrest Gump"), (2, "Forrest Gump")]
 - Second list – List of liked directors (userId, likedDirector)
 - [(1, "Woody Allen"), (2, "Quentin Tarantino"), (2, "Alfred Hitchcock")]

Cogroup transformation: Example

- Create a new RDD of pairs containing one pair for each userId (key) associated with
 - The list of liked movies
 - The list of liked directors

Cogroup transformation: Example

■ Inputs

- [(1, "Star Trek"), (1, "Forrest Gump"), (2, "Forrest Gump")]
- [(1, "Woody Allen"), (2, "Quentin Tarantino"), (2, "Alfred Hitchcock")]

■ Output

- (1, (["Star Trek", "Forrest Gump"], ["Woody Allen"]))
- (2, (["Forrest Gump"], ["Quentin Tarantino", "Alfred Hitchcock"]))

Cogroup transformation: Example

```
# Create the first local python list
movies= [(1, "Star Trek"), (1, "Forrest Gump"), (2, "Forrest Gump")]
```

```
# Create the RDD of pairs from the first local list
moviesPairRDD = sc.parallelize(movies)
```

```
# Create the second local python list
directors = [ (1, "Woody Allen"), (2, "Quentin Tarantino"), \
              (2, "Alfred Hitchcock")]
```

```
# Create the RDD of pairs from the second local list
directorsPairRDD = sc.parallelize(directors)
```

```
# Cogroup movies and directors per user
cogroupPairRDD = moviesPairRDD.cogroup(directorsPairRDD)
```

Cogroup transformation: Example

```
# Create the first local python list
```

```
movies= [(1, "Star Trek"), (1, "Forrest Gump"), (2, "Forrest Gump")]
```

```
# Create the RDD of pairs from the first local list
```

```
moviesPairRDD = sc.parallelize(movies)
```

```
# Create the second local python list
```

```
directors = [ (1, "Woody Allen"), (2, "Quentin Tarantino"), \  
              (2, "Alfred Hitchcock")]
```

Note that the value part of the returned tuples is a tuple containing two "lists":

- The first value contains the "list" of movies (iterable) liked by a user
- The second value contains the "list" of directors (iterable) liked by a user

```
# Cogroup movies and directors per user
```

```
cogroupPairRDD = moviesPairRDD.cogroup(directorsPairRDD)
```

Transformations on two RDDs of key-value pairs: Summary

Transformations on two RDDs of key-value pairs: Summary

- All the examples reported in the following tables are applied on the following two RDDs of key-value pairs
 - inputRDD1: [('k1', 2), ('k3', 4), ('k3', 6)]
 - inputRDD2: [('k3', 9)]

Transformations on two RDDs of key-value pairs: Summary

Transformation	Purpose	Example	Result
<code>subtractByKey(other)</code>	Return a new RDD of key-value pairs. The returned pairs are those of input RDD on which the method is invoked such that the key part does not occur in the keys of the RDD that is passed as parameter. The values are not considered to take the decision.	<code>inputRDD1.subtractByKey(inputRDD2)</code>	<code>[('k1',2)]</code>
<code>join(other)</code>	Return a new RDD of pairs corresponding to join of the two input RDDs. The join is based on the value of the key.	<code>inputRDD1.join(inputRDD2)</code>	<code>[('k3', (4,9)), ('k3', (6,9))]</code>

Transformations on two RDDs of key-value pairs: Summary

Transformation	Purpose	Example	Result
cogroup(other)	<p>For each key k in one of the two input RDDs of pairs, return a pair (k, tuple), where tuple contains:</p> <ul style="list-style-type: none">- the list (iterable) of values of the first input RDD associated with key k- the list (iterable) of values of the second input RDD associated with key k	<pre>inputRDD1. cogroup (inputRDD2)</pre>	<pre>[('k1', ([2], [])) , ('k3', ([4, 6], [9]))]</pre>

Actions on RDDs of key-value pairs

Actions on RDDs of key-value pairs

- Spark supports also some specific actions on RDDs of key-value pairs
 - `countByKey`, `collectAsMap`, `lookup`

CountByKey action

CountByKey action

- Goal
 - The countByKey action returns a local python dictionary containing the information about the number of elements associated with each key in the input RDD of key-value pairs
 - i.e., the number of times each key occurs in the input RDD
 - **Pay attention to the number of distinct keys of the input RDD of pairs**
 - **If the number of distinct keys is large, the result of the action cannot be stored in a local variable of the Driver**

CountByKey action

- Method
 - The countByKey action is based on the `countByKey()` method of the `RDD` class

CountByKey action: Example 1

- Create an RDD of pairs from the following python list
 - [("Forrest Gump", 4), ("Star Trek", 5), ("Forrest Gump", 3)]
 - Each pair contains a movie and the rating given by someone to that movie
- Compute the number of ratings for each movie

CountByKey action: Example 1

```
# Create the local python list
movieRating= [("Forrest Gump", 4), ("Star Trek", 5), ("Forrest Gump", 3)]

# Create the RDD of pairs from the local collection
movieRatingRDD = sc.parallelize(movieRating)

# Compute the number of rating for each movie
movieNumRatings = movieRatingRDD.countByKey()

# Print the result on the standard output
print(movieNumRatings)
```

CountByKey action: Example 1

```
# Create the local python list
```

```
movieRating= [("Forrest Gump", 4), ("Star Trek", 5), ("Forrest Gump", 3)]
```

```
# Create the RDD of pairs from the local collection
```

```
movieRatingRDD = sc.parallelize(movieRating)
```

```
# Compute the number of rating for each movie
```

```
movieNumRatings= movieRatingRDD.countByKey()
```

Pay attention to the size of the returned local python dictionary (i.e., the number of distinct movies in this case).

CollectAsMap action

CollectAsMap action

- Goal
 - The collectAsMap action returns a local dictionary containing the same pairs of the considered input RDD of pairs
 - **Pay attention to the size of the returned RDD**
- Method
 - The collectAsMap action is based on the **collectAsMap()** method of the **RDD** class

CollectAsMap action

- **Pay attention** that the **collectAsMap** action **returns a dictionary** object
- **A dictionary cannot contain duplicate keys**
 - Each key can be associated with at most one value
 - If the “input” RDD of pairs contains more than one pair with the same key, only one of those pairs is stored in the returned local python dictionary
 - Usually, the last one occurring in the input RDD of pairs
- Use `collectAsMap` only if you are sure that each key appears only once in the input RDD of key-value pairs

CollectAsMap vs collect

- The collectAsMap() action returns a local dictionary while collect() return a list of key-value pairs (i.e., a list of tuples)
 - The list of pairs returned by collect() can contain more than one pair associated with the same key

CollectAsMap action: Example 1

- Create an RDD of pairs from the following python list
 - [("User1", "Paolo"), ("User2", "Luca"), ("User3", "Daniele")]
 - Each pair contains a userId and the name of the user
- Retrieve the pairs of the created RDD of pairs and store them in a local python dictionary that is instantiated in the Driver

CollectAsMap action: Example 1

```
# Create the local python list
users = [("User1", "Paolo"), ("User2", "Luca"), ("User3", "Daniele")]

# Create the RDD of pairs from the local list
usersRDD = sc.parallelize(users)

# Retrieve the content of usersRDD and store it in a
# local python dictionary
retrievedPairs = usersRDD.collectAsMap()

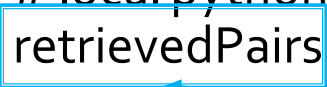
# Print the result on the standard output
print(retrievedPairs)
```

CollectAsMap action: Example 1

```
# Create the local python list
users = [("User1", "Paolo"), ("User2", "Luca"), ("User3", "Daniele")]

# Create the RDD of pairs from the local list
usersRDD = sc.parallelize(users)

# Retrieve the content of usersRDD and store it in a
# local python dictionary
retrievedPairs = usersRDD.collectAsMap()
```



Pay attention to the size of the returned local python dictionary (i.e., the number of distinct users in this case).

Lookup action

Lookup action

- Goal
 - The lookup(**k**) action returns a local python list containing the values of the pairs of the input RDD associated with the key **k** specified as parameter
- Method
 - The lookup action is based on the **lookup(key)** method of the **RDD** class

Lookup action: Example 1

- Create an RDD of pairs from the following python list
 - [("Forrest Gump", 4), ("Star Trek", 5), ("Forrest Gump", 3)]
 - Each pair contains a movie and the rating given by someone to that movie
- Retrieve the ratings associated with the movie "Forrest Gump" and store them in a local python list in the Driver

Lookup action: Example 1

```
# Create the local python list
movieRating= [("Forrest Gump", 4), ("Star Trek", 5), ("Forrest Gump", 3)]

# Create the RDD of pairs from the local collection
movieRatingRDD = sc.parallelize(movieRating)

# Select the ratings associated with "Forrest Gump"
movieRatings = movieRatingRDD.lookup("Forrest Gump")

# Print the result on the standard output
print(movieRatings)
```

Lookup action: Example 1

```
# Create the local python list
```

```
movieRating= [("Forrest Gump", 4), ("Star Trek", 5), ("Forrest Gump", 3)]
```

```
# Create the RDD of pairs from the local collection
```

```
movieRatingRDD = sc.parallelize(movieRating)
```

```
# Select the ratings associated with "Forrest Gump"
```

```
movieRatings = movieRatingRDD.lookup("Forrest Gump")
```

Pay attention to the size of the returned list (i.e., the number of ratings associated with "Forrest Gump" in this case).

Actions on RDDs of key-value pairs: Summary

Actions on RDDs of key-value pairs: Summary

- All the examples reported in the following tables are applied on the following RDD of key-value pairs
 - inputRDD: [('k1', 2), ('k3', 4), ('k3', 6)]

Actions on RDDs of key-value pairs:

Summary

Transformation	Purpose	Example	Result
<code>countByKey()</code>	Return a local python dictionary containing the number of elements in the input RDD for each key of the input RDD of pairs	<code>inputRDD.countByKey()</code>	<code>{('k1',1), ('K3',2)}</code>
<code>collectAsMap()</code>	Return a local python dictionary containing the pairs of the input RDD of pairs	<code>inputRDD.collectAsMap()</code>	<code>{('k1', 2), ('k3', 6)}</code> Or <code>{('k1', 2), ('k3', 4)}</code> Depending on the order of the pairs in the input RDD of pairs
<code>lookup(key)</code>	Return a local python list containing all the values associated with the key specified as parameter	<code>inputRDD.lookup('k3')</code>	<code>[4, 6]</code>