Big data: architectures and data analytics

RDD-based programming

RDDs and key-value pairs

Transformations on pairs of Pair RDDs

- Spark supports also some transformations on two PairRDDs
  - SubtractByKey, join, coGroup, etc.

SubtractByKey transformation

SubtractByKey transformation

- Goal
  - Create a new PairRDD containing only the pairs of the input PairRDD associated with a key that is not appearing as key in any pair of the other PairRDD
    - The data type of the new PairRDD is the same of the "input" PairRDD
    - The input PairRDD and the other PairRDD must have the same type of keys

6

SubtractByKey transformation

- Method
  - The subtractByKey transformation is based on the **JavaPairRDD<K,V> subtractByKey(JavaPairRDD<K,U> other)** method of the **JavaPairRDD<K,V>** class

7

SubtractByKey transformation: Example

- Create two JavaPairRDD from two local Java 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 PairRDD containing only the profiles of the non-banned users

8

SubtractByKey transformation: Example

```
// Create the first local Java collection
ArrayList<Tuple2<String, Integer>> profiles =
                    new ArrayList<Tuple2<String, Integer>>();

Tuple2<String, Integer> localPair;
localPair = new Tuple2<String, Integer>("PaoloG", 40);
profiles.add(localPair);

localPair = new Tuple2<String, Integer>("Giorgio", 22);
profiles.add(localPair);

localPair = new Tuple2<String, Integer>("PaoloB", 35);
profiles.add(localPair);

// Create the JavaPairRDD from the local collection
JavaPairRDD<String, Integer> profilesPairRDD = sc.parallelizePairs(profiles);
```

9

SubtractByKey transformation: Example

```
// Create the second local Java collection
ArrayList<Tuple2<String, String>> banned =
                    new ArrayList<Tuple2<String, String>>();
Tuple2<String, String> localPair2;
localPair2 = new Tuple2<String, String>("PaoloB", "spam");
banned.add(localPair2);

localPair2 = new Tuple2<String, String> ("Giorgio", "Vandalism");
banned.add(localPair2);
// Create the JavaPairRDD from the local collection
JavaPairRDD<String, String> bannedPairRDD = sc.parallelizePairs(banned);

// Select the profiles of the "good" users
JavaPairRDD<String, Integer> selectedUsersPairRDD =
                    profilesPairRDD.subtractByKey(bannedPairRDD);
```

10

Join transformation

Join transformation

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

12

Join transformation

- Method
  - The join transformation is based on the **JavaPairRDD <K, Tuple2<V,U>> join(JavaPairRDD<K,U>)** method of the **JavaPairRDD<K,V>** class

13

Join transformation: Example

- Create two JavaPairRDD from two local Java 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 PairRDD associating each question with its answers
  - One pair for each possible pair question - answer

14

Join transformation: Example

```
// Create the first local Java collection
ArrayList<Tuple2<Integer, String>> questions=
                    new ArrayList<Tuple2<Integer, String>>();

Tuple2<Integer, String> localPair;
localPair = new Tuple2<Integer, String>(1, "What is .. ?");
questions.add(localPair);


localPair = new Tuple2<Integer, String> (2, "Who is ..?");
questions.add(localPair);

// Create the JavaPairRDD from the local collection
JavaPairRDD<Integer, String> questionsPairRDD =
                                    sc.parallelizePairs(questions);
```

15

Join transformation: Example

```
// Create the second local Java collection
ArrayList<Tuple2<Integer, String>> answers =
                    new ArrayList<Tuple2<Integer, String>>();
Tuple2<Integer, String> localPair2;
localPair2 = new Tuple2<Integer, String>(1, "It is a car");
answers.add(localPair2);

localPair2 = new Tuple2<Integer, String>(1, "It is a byke");
answers.add(localPair2);

localPair2 = new Tuple2<Integer, String>(2, "She is Jenny");
answers.add(localPair2);

// Create the JavaPairRDD from the local collection
JavaPairRDD<Integer, String> answersPairRDD=sc.parallelizePairs(answers);
```

16

Join transformation: Example

```
// Join questions with answers
JavaPairRDD<Integer, Tuple2<String, String>> joinPairRDD =
                questionsPairRDD.join(answersPairRDD);
```

17

CoGroup transformation

Cogroup transformation

- Goal
  - Associated each key **k** of the input PairRDDs with
    - The list of values associated with **k** in the input PairRDD
    - And the list of values associated with **k** in the other PairRDD
  - The new PairRDD
    - Has the same key data type of the "input" PairRDDs
    - Has a tuple as value (the two lists of values of the two input pairs)
    - The input PairRDD and the other PairRDD
      - Must have the same type of keys
      - But the data types of the values can be different

19

Cogroup transformation

- Method
  - The cogroup transformation is based on the **JavaPairRDD <K, Tuple2<Iterable<V>, Iterable<U>>> cogroup(JavaPairRDD<K,U>)** method of the **JavaPairRDD<K,V>** class

20

Cogroup transformation: Example

- Create two JavaPairRDD from two local Java 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")}
- Create a new PairRDD containing one pair for each userId (key) associated with
  - The list of liked movies
  - The list of liked directors

21

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"]))

22

Cogroup transformation: Example

```
// Create the first local Java collection
ArrayList<Tuple2<Integer, String>> movies=
                    new ArrayList<Tuple2<Integer, String>>();

Tuple2<Integer, String> localPair;
localPair = new Tuple2<Integer, String>(1, "Star Trek");
movies.add(localPair);

localPair = new Tuple2<Integer, String>(1, "Forrest Gump");
movies.add(localPair);

localPair = new Tuple2<Integer, String>(2, "Forrest Gump");
movies.add(localPair);

// Create the JavaPairRDD from the local collection
JavaPairRDD<Integer, String> moviesPairRDD = sc.parallelizePairs(movies);
```

23

Cogroup transformation: Example

```
// Create the second local Java collection
ArrayList<Tuple2<Integer, String>> directors =
                    new ArrayList<Tuple2<Integer, String>>();
Tuple2<Integer, String> localPair2;
localPair2 = new Tuple2<Integer, String>(1, "Woody Allen");
directors.add(localPair2);

localPair2 = new Tuple2<Integer, String>(2, "Quentin Tarantino");
directors.add(localPair2);

localPair2 = new Tuple2<Integer, String>(2, "Alfred Hitchcock");
directors.add(localPair2);

// Create the JavaPairRDD from the local collection
JavaPairRDD<Integer, String> directorsPairRDD=
                    sc.parallelizePairs(directors);
```

24

Cogroup transformation: Example

```
// Cogroup movies and directors per user
JavaPairRDD<Integer, Tuple2<Iterable<String>, Iterable<String>>>
        cogroupPairRDD = moviesPairRDD.cogroup(directorsPairRDD);
```

25

Transformations on two PairRDDs: Summary

Transformations on two Pair RDDs: Summary

- All the examples reported in the following tables are applied on the following two PairRDDs
  - inputRDD1 {("k1", 2), ("k3", 4), ("k3", 6)}
  - inputRDD2 {("k3", 9)}

27

Transformations on two Pair RDDs: Summary

Transformation	Purpose	Example	Result
JavaPairRDD<K,V> subtractByKey(JavaPairRDD<K,U>)	Return a new PairRDD where the pairs associated with a key appearing only in the "input"" PairRDD and not in the one passed as parameter. The values are not considered to take the decision.	inputRDD1. subtract (inputRDD2)	{("K1",2)}
JavaPairRDD <K, Tuple2<V,U>> join(JavaPairRDD<K,U>)	Return a new PairRDD corresponding to join of the two PairRDDs. The join is based in the value of the key.	inputRDD1.join (inputRDD2)	{("k3", (4,9)), ("k3", (6,9))}

28

Transformations on two Pair RDDs: Summary

Transformation	Purpose	Example	Result
JavaPairRDD <K, Tuple2<Iterable<V>, Iterable<U>>> cogroup(JavaPairRDD<K,U>)	For each key k in one of the two PairRDDs, return a pair (k, tuple), where tuple contains the list of values of the first PairRDD with key k in the first element of the tuple and the list of values of the second PairRDD with key k in the second element of the tuple.	inputRDD1. cogroup (inputRDD2)	{("K1", ([2], [])), ("K3", ([4, 6], [9]))}

29

Actions on Pair RDDs

Actions on Pair RDDs

- Spark supports also some specific actions on PairRDDs
  - countByKey, collectAsMap, lookup

31

CountByKey action

CountByKey action

- Goal
  - The countByKey action returns a local Java Map object containing the information about the number of elements associated with each key in the PairRDD
    - i.e., the number of times each key occurs in the PairRDD
  - **Pay attention to number of distinct keys of the PairRDD**
  - **If the number of distinct keys is large, the result of the action cannot be memorized in a local variable of the Driver**

33

CountByKey action

- Method
  - The countBykey action is based on the **java.util.Map<K, java.lang.Object> countBykey()** method of the **JavaPairRDD<K,V>** class
    - The values of the returned java.util.Map are returned as "generic" java.lang.Object
    - However, they are java.lang.Long objects

34

CountByKey action: Example 1

- Create a JavaPairRDD from the following Java list
  - {("Forrest Gump", 4), ("Star Trek", 5) , ("Forrest Gump", 3)}
  - Each pair contains a movie and the rating given by someone to the movie
- Compute the number of ratings for each movie

35

CountByKey action: Example 1

```
// Create the local Java collection
ArrayList<Tuple2<String , Integer>> movieRating=
                    new ArrayList<Tuple2<String, Integer>>();

Tuple2<String, Integer> localPair;
localPair = new Tuple2<String, Integer>("Forrest Gump", 4);
movieRating.add(localPair);

localPair = new Tuple2<String, Integer>("Star Trek", 5);
movieRating.add(localPair);

localPair = new Tuple2<String, Integer>("Forrest Gump", 3);
movieRating.add(localPair);
```

36

CountByKey action: Example 1

```
// Create the JavaPairRDD from the local collection
JavaPairRDD< String, Integer> movieRatingRDD =
                                    sc.parallelizePairs(movieRating);

// Compute the number of rating for each movie
 java.util.Map<String, java.lang.Object> movieNumRatings =
                                    movieRatingRDD.countByKey();

// Print the result on the standard output
System.out.println(movieNumRatings);
```

37

CountByKey action: Example 1

```
// Create the JavaPairRDD from the local collection
JavaPairRDD< String, Integer> movieRatingRDD =
                                    sc.parallelizePairs(movieRating);

// Compute the number of rating for each movie
 java.util.Map<String, java.lang.Object> movieNumRatings =
                                    movieRatingRDD.countByKey();

// Pr
Syst
```

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

38

CollectAsMap action

CollectAsMap action

- Goal
  - The collectAsMap action returns a local Java java.util.Map<K,V> containing the same pairs of the considered PairRDD
  - **Pay attention to the size of the PairRDD**
- Method
  - The collectAsMap action is based on the **java.util.Map<K,V> collectAsMap()** method of the **JavaPairRDD<K,V>** class

40

CollectAsMap action

- **Pay attention** that the **collectAsMap** action **returns a java.util.Map** object
- **A Map cannot contain duplicate keys**
  - Each key can map to at most one value
  - If the "input" PairRDD contains more than one pair with the same key, only one of those pairs is stored in the returned local Java Map
    - The last one in the PairRDD
- Use collectAsMap only if you are sure that each key appears only once in the PairRDD

41

CollectAsMap action: Example 1

- Create a JavaPairRDD from the following Java list
  - {("User1", "Paolo"), ("User2", "Luca"), ("User3", "Daniele")}
  - Each pair contains a userId and the name of the user
- Retrieve the pairs of the created PairRDD and store them in a local Java Map that is instantiated in the Driver

42

CollectAsMap action: Example 1

```
// Create the local Java collection
ArrayList<Tuple2<String , String>> users=
                    new ArrayList<Tuple2<String , String>>();

Tuple2<String , String> localPair;
localPair = new Tuple2<String , String>("User1", "Paolo");
users.add(localPair);

localPair = new Tuple2<String , String>("User2", "Luca");
users.add(localPair);

localPair = new Tuple2<String , String>("User3", "Daniele");
users.add(localPair);
```

43

CollectAsMap action: Example 1

```
// Create the JavaPairRDD from the local collection
JavaPairRDD< String, String> usersRDD =
                    sc.parallelizePairs(users);


// Retrieve the content of usersRDD and store it in a local Java Map
 java.util.Map<String, String> retrievedPairs = usersRDD.collectAsMap();

// Print the result on the standard output
System.out.println(retrievedPairs);
```

44

CollectAsMap action: Example 1

```
// Create the JavaPairRDD from the local collection
JavaPairRDD< String, String> usersRDD =
                    sc.parallelizePairs(users);


// Retrieve the content of usersRDD and store it in a local Java Map
java.util.Map<String, String> retrievedPairs = usersRDD.collectAsMap();

// Print the
System.out.println(retrievedPairs);
```

Pay attention to the size of the returned map

45

Lookup action

Lookup action

- Goal
  - The lookup(**k**) action returns a local Java java.util.List<V> containing the values of the pairs of the PairRDD associated with the key **k** specified as parameter
- Method
  - The lookup action is based on the **java.util.List<V> lookup(K key)** method of the **JavaPairRDD<K,V>** class

47

Lookup action: Example 1

- Create a JavaPairRDD from the following Java list
  - {("Forrest Gump", 4), ("Star Trek", 5) , ("Forrest Gump", 3)}
  - Each pair contains a movie and the rating given by someone to the movie
- Retrieve the ratings associated with the movie "Forrest Gump" and store them in a local Java list in the Driver

48

Lookup action: Example 1

```
// Create the local Java collection
ArrayList<Tuple2<String , Integer>> movieRating=
                    new ArrayList<Tuple2<String, Integer>>();

Tuple2<String, Integer> localPair;
localPair = new Tuple2<String, Integer>("Forrest Gump", 4);
movieRating.add(localPair);

localPair = new Tuple2<String, Integer>("Star Trek", 5);
movieRating.add(localPair);

localPair = new Tuple2<String, Integer>("Forrest Gump", 3);
movieRating.add(localPair);
```

49

Lookup action: Example 1

```
// Create the JavaPairRDD from the local collection
JavaPairRDD< String, Integer> movieRatingRDD =
                    sc.parallelizePairs(movieRating);


// Select the ratings associated with "Forrest Gump"
 java.util.List<Integer> movieRatings =
                    movieRatingRDD.lookup("Forrest Gump");

// Print the result on the standard output
System.out.println(movieRatings);
```

50

Lookup action: Example 1

```
// Create the JavaPairRDD from the local collection
JavaPairRDD< String, Integer> movieRatingRDD =
                                    sc.parallelizePairs(movieRating);


// Select the ratings associated with "Forrest Gump"
java.util.List<Integer> movieRatings =
                                    movieRatingRDD.lookup("Forrest Gump");

// Pr
Syst
```

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

51

Actions on PairRDDs: Summary

Actions on PairRDDs: Summary

- All the examples reported in the following tables are applied on the following PairRDD
  - {("k1", 2), ("k3", 4), ("k3", 6)}

53

Actions on PairRDDs: Summary

Transformation	Purpose	Example	Result
java.util.Map<K,java.lang.Object> countByKey()	Return a local Java java.util.Map containing the number of elements in the input PairRDD for each key of the input PairRDD.	inputRDD.countByKey()	{("K1",1), ("K3",2)}
java.util.Map<K,V> collectAsMap()	Return a local Java java.util.Map containing the pairs of the input PairRDD	inputRDD.collectAsMap()	{("k1", 2), ("k3", 6)} Or {("k1", 2), ("k3", 4)} Depending on the order of the pairs in the PairRDD

54

Actions on PairRDDs: Summary

Transformation	Purpose	Example	Result
java.util.List<V> lookup(K key)	Return a local Java java.util.List containing all the values associated with the key specified as parameter	inputRDD. lookup("k3")	{4, 6}

55