

# CS 378 – Big Data Programming

Lecture 24

RDDs

# Review

- Assignment 10 – Download and run Spark
  - WordCount implementation
  - WordCount – alternative implementation

# Basic RDD

- Transformations we've discussed
  - `filter(function)`
  - `map(function)`
  - `flatMap(function)`
  - `mapToPair(function)`
- How do these work for multiple partitions?
- Is a shuffle over the network required?

# Spark Transformations

- Filter

- Apply a function to each element of an RDD, return those elements that evaluate to true
- Source RDD element type: `T`
- Result RDD element type: `T`
- Java function (class) type: `Function<T, Boolean>`
- Java method: `Boolean call(T t)`

# Spark Transformations

- Map
  - Apply a function to each element of an RDD, return the result of applying the function
  - RDD element type:  $T$
  - Result RDD element type:  $R$
  - Java function (class) type: `Function<T, R>`
  - Java method: `R call(T t)`

# Spark Transformations

- Flat Map
  - Apply a function to each element of an RDD, return the result of applying the function (an Iterable)
  - Source RDD element type: `T`
  - Result RDD element type: `R`
  - Java function (class) type: `FlatMapFunction<T, R>`
  - Java method: `Iterable<R> call(T t)`

# Spark Transformations

- Map to Pair
  - Apply a function to each element of an RDD, return the result of applying the function (a key/value pair)
  - Source RDD element type: `T`
  - Result RDD element type: `<K, V>`
  - Java function (class) type: `PairFunction<T, K, V>`
  - Java method: `Tuple2<K, V> call(T t)`

# Basic RDD

- Additional transformations

- `distinct()`

- `union(otherRDD)`

- `intersection(otherRDD)`

- `subtract(otherRDD)`

- `cartesian(otherRDD)`

- `sample(withReplacement, fraction, [seed])`



# Spark Transformations

- Distinct
  - Remove duplicates
  - Source RDD element type:  $T$
  - Result RDD element type:  $T$

# Spark Transformations

- Union
  - Compute the union of two RDDs
  - Duplicates are not removed
  - Source RDD element type:  $T$
  - “Other” RDD element type:  $T$
  - Result RDD element type:  $T$

# Spark Transformations

- Intersection
  - Compute the intersection of two RDDs
  - Source RDD element type:  $T$
  - “Other” RDD element type:  $T$
  - Result RDD element type:  $T$
  - Requires a shuffle over the network. Why?
  - Does union require a shuffle?

# Spark Transformations

- Subtract
  - Remove the elements of one RDD from another
  - Source RDD element type:  $T$
  - “Other” RDD element type:  $T$
  - Result RDD element type:  $T$
  - Require a shuffle over the network?

# Spark Transformations

- Cartesian
  - Compute the cross product (Cartesian product) of two RDDs
  - Source RDD element type: T
  - “Other” RDD element type: U
  - Result RDD element type: `JavaPairRDD<T, U>`
  - Require a shuffle over the network?
  - Very expensive in time and space

# Spark Transformations

- Sample
  - Sample from an RDD
  - Source RDD element type: T
  - Result RDD element type: T
  - `sample(withReplacement, fraction, [seed])`
  - Require a shuffle over the network?

# Basic RDD

- Actions we've discussed
  - `collect()`
  - `count()`
  - `take(n)`
  - `first()`
  - `saveAsTextFile()`

# Basic RDD

- Additional actions
  - `countByValue()`
    - Returns a map from RDD element to count (integer)
    - Could we use this for WordCount?
  - `takeOrdered(n, comparator)`
    - Sort the RDD elements, then `take()`
  - `takeSample(withReplacement, num)`
    - Start sampling to get `num` elements



# Basic RDD

- Additional actions

- `reduce (Function2<T, T, T> f)`

- Returns an object of type `T`

- The function `f` should be

- commutative

- Associative

- Does summing counts meet the specs for `f` ?

- Does concatenating strings meet the specs for `f` ?

# Assignment 11

- Inverted index, implemented in Spark
- We'll use the same data (Assignment 3)
  - Remove punctuation
  - For each word, output a list of verses containing the word, in sorted order