Extreme Computing

Distributed Data-Parallel Programming

THE UNIVERSITY *of* EDINBURGH

Amir Shaikhha, Fall 2023

Part 4

Distributed Collections in Spark

Resilient Distributed Datasets (RDDs)

- Inspired by immutable Scala collections
- Most operations are higher-order functions

```
abstract class RDD[T] {
  def map[U](f: T => U): RDD[U] = ...
  def flatMap[U](f: T => TraversableOnce[U]): RDD[U] = ...
  def filter(p: T => Boolean): RDD[T] = ...
  def reduce(f: (T, T) => T): T = ...
}
```

Scala Collections vs. Spark RDD

Scala collection	Spark RDD
map	**map**
flatMap	**flatMap**
filter	**filter**
reduce	**reduce**
fold	**fold**

```
map[B](f: A=> B): List[B] // Scala List
map[B](f: A=> B): RDD[B] // Spark RDD

flatMap[B](f: A=> TraversableOnce[B]): List[B]// Scala List
flatMap[B](f: A=> TraversableOnce[B]): RDD[B]// Spark RDD
```

Scala Collections vs. Spark RDD (cont.)

- Similarity: API and usage
- Difference: Data is distributed

```
val lines: List[String]
lines.filter(line => line.contains("Amir")).count()

val lines: RDD[String]
lines.filter(line => line.contains("Amir")).count()
```

RDD Creation (1)

Transformation on an existing RDD

```
abstract class RDD[T] {
  def map[U](f: T => U): RDD[U] = ...
  def flatMap[U](f: T => TraversableOnce[U]): RDD[U] = ...
  def filter(p: T => Boolean): RDD[T] = ...
}
```

RDD Creation (2)

From a `SparkContext` object

- Connection between the Spark cluster and application
- Methods for creating & populating a new RDD

```
class SparkContext {
  def textFile(path: String): RDD[String] = ...
  def parallelize[T](seq: Seq[T]): RDD[T] = ...
}
```

Example: Word count

- A.K.A. the “Hello world!” of large-scale programming

SparkContext

```
// Create an RDD
val lines: RDD[String] = sc.textFile("hdfs:// ...")

// separate lines into words
val words = lines.flatMap(line => line.split(" "))

// include something to count
val counts = words.map(word => (word, 1))

// sum up the 1s in the pairs
val count = counts.reduceByKey(_ + _)
```

Transformers vs. Actions

Transformers

- Return new collections as results
  - Not single values
  - E.g., `map`, `filter`, `flatMap`, `groupBy`

Actions

- Compute a result using an RDD
  - Return the result or store externally

Lazy Evaluation in Spark

- Transformers are lazy
  - The result IS NOT immediately computed
- Actions are eager
  - The result IS immediately computed

Laziness/eagerness enables the programmer to control the network communication

Common Transformations

Spark RDD	Signature / Description
`map`	`map[U](f: T => U): RDD[U]` Apply a function to each element in the RDD and return an RDD of the result.
`flatMap`	`flatMap[U](f: T => TraversableOnce[U]): RDD[U]` Apply a function to each element in the RDD and return an RDD of the contents of the iterators returned.
`filter`	`filter(p: T => Boolean): RDD[T]` Apply predicate function to each element in the RDD and return an RDD of elements that have passed the predicate condition.
`distinct`	`distinct(): RDD[T]` Return RDD with duplicates removed.

Common Actions

Spark RDD	Signature / Description
`collect`	`collect() : Array[T]` Return all elements from RDD.
`count`	`count() : Long` Return the number of elements in the RDD.
`take`	`take(num: Int): Array[T]` Return the first `num` elements of the RDD.
`reduce`	`reduce(op: (T, T) => T): T` Combine the elements in the RDD together using `op` function and return result.

Example: Word Count

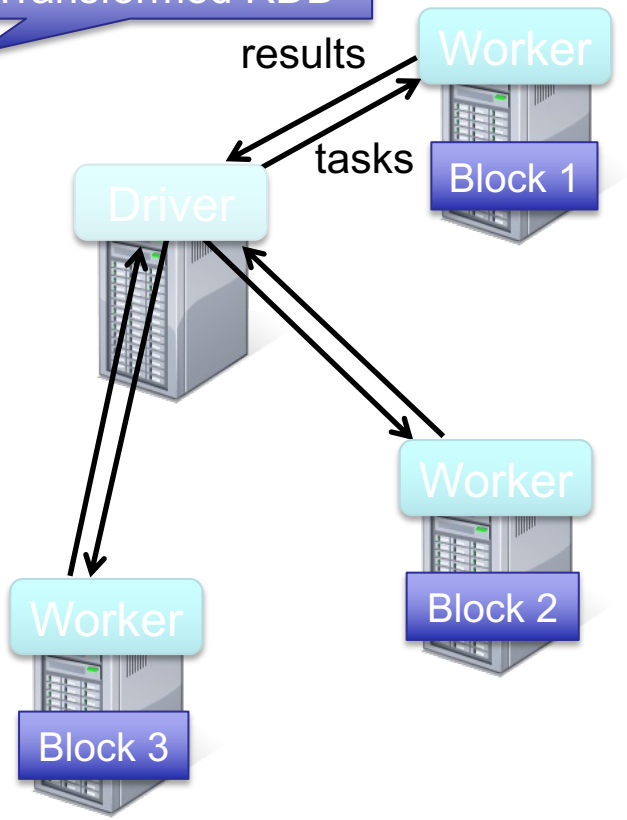
```
lines = sc.textFile(“hdfs://...”)
words = lines.flatMap(line => line.split(" "))
wordOne = words.map(word => (word, 1))
wordCount = wordOne.reduceByKey(_ + _)

wordCount.collect()
```

Base RDD

Transformed RDD

Action



QUESTIONS?