Extreme Computing

Distributed Data-Parallel Programming

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

<table>
<thead>
<tr>
<th>Scala collection</th>
<th>Spark RDD</th>
</tr>
</thead>
<tbody>
<tr>
<td>map</td>
<td>map</td>
</tr>
<tr>
<td>flatMap</td>
<td>flatMap</td>
</tr>
<tr>
<td>filter</td>
<td>filter</td>
</tr>
<tr>
<td>reduce</td>
<td>reduce</td>
</tr>
<tr>
<td>fold</td>
<td>fold</td>
</tr>
</tbody>
</table>

`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

```scala
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

```scala
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

```scala
// 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

<table>
<thead>
<tr>
<th>Spark RDD</th>
<th>Signature / Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>map</strong></td>
<td><code>map[U](f: T =&gt; U): RDD[U]</code>  &lt;br&gt;Apply a function to each element in the RDD and return an RDD of the result.</td>
</tr>
<tr>
<td><strong>flatMap</strong></td>
<td><code>flatMap[U](f: T =&gt; TraversableOnce[U]): RDD[U]</code>  &lt;br&gt;Apply a function to each element in the RDD and return an RDD of the contents of the iterators returned.</td>
</tr>
<tr>
<td><strong>filter</strong></td>
<td><code>filter(p: T =&gt; Boolean): RDD[T]</code>  &lt;br&gt;Apply predicate function to each element in the RDD and return an RDD of elements that have passed the predicate condition.</td>
</tr>
<tr>
<td><strong>distinct</strong></td>
<td><code>distinct(): RDD[T]</code>  &lt;br&gt;Return RDD with duplicates removed.</td>
</tr>
</tbody>
</table>
## Common Actions

<table>
<thead>
<tr>
<th>Spark RDD</th>
<th>Signature / Description</th>
</tr>
</thead>
</table>
| **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

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

Diagram:
- **Base RDD**
- **Transformed RDD**
- **Driver**
- **Worker**
- **Block 1**
- **Block 2**
- **Block 3**

Arrow connections show:
- **tasks** from Driver to Workers
- **results** from Workers to Driver
QUESTIONS?