Lab 3.2: Operations On Multiple RDDs

Overview

In this lab, we will work with somewhat more complex operations that operate on multiple RDDs. We will continue to use very small data sets in order to focus on gaining familiarity with the Spark API.

Builds on

None

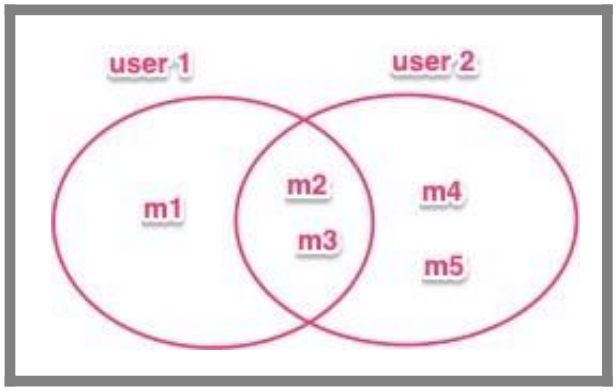
Run time

30-40 minutes

Meetups: Our data for this lab

For this lab, assume that we have users who are attending several meetups. We will start with two users, who are attending the following meetups:

- User1 attends meetups: m1, m2 and m3. User2
- attends meetups: m2, m3, m4 and m5.



Each user's meetups will be in separate RDDs, so you'll have two RDDs to work with. We'll analyze the data for the two users by performing operations over both RDDs.
We'll also look at the Spark Shell UI to get an idea of how Spark is processing the data.

Operations that you work with will include `union` , `intersection` , `distinct`, and `subtract`.

Create RDDs with our Data

Tasks

- Go to the spark shell.
- In the shell, create two RDDs holding the meetups for each user. Keep it
- simple, and parallelize a list as illustrated below.
- Once you've done that, look at the Jobs tab of the UI - do you see anything?
    - Why or why not?

```
// Scala
val u1 = sc.parallelize(List("m1", "m2", "m3"))
val u2 = sc.parallelize(???)
```

```
# Python
u1 = sc.parallelize(["m1", "m2", "m3"])
u2 = sc.parallelize(???)
```

Perform Operations to Analyze Data

Tasks

- Using your two RDDs, find meetups **common to both users**
    - What operation will you use for this?
    - Look at the Jobs tab in the UI. Do you see something now? (You should.)
    - Go to the detail for this new job (click through in the Description column), and look at the DAG Visualization. We'll discuss it later, just look for now.
    - You can click on one of the blue boxes in the diagram (a stage) to get more detail on that stage (we'll look at the stages again later in the lab).

- Find meetups attended by **either user1 or user2**.
    - How do you do this? What if there are duplicates in the results? How would you remove them?
    - Look at its Job DAG after you've done this transformation.

- Find meetups that **ONLY u1 attended**.

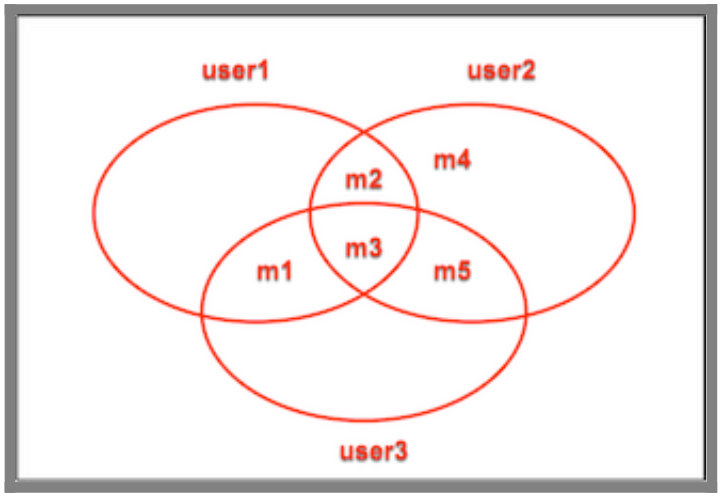
    - That is, u1 attended, but u2 did not.
    - Look at the job DAG in the UI once you've done this.

- Create two sets of meetup recommendations, where **each user recommends the meetups the other did NOT attend**. The results should be:
    - u1 will recommend to u2: m1
    - u2 will recommend to u1: m4, m5
    - How can you do this - what operation(s) are needed? Look at
    - the job DAG in the UI once you've done this.

Recommendations (With Three Users)

Let's introduce user3

- User3 attends meetups: m1, m3, and m5, as illustrated below.



Consider how to make recommendations, based on the following (very simple) requirements.

- A user should not be attending a meetup to be recommended to them.
- A meetup should be attended by both of the other users to be recommended.

Based on this, we would recommend the following to the users: u1: m5

- u2: M1
- u3: M2

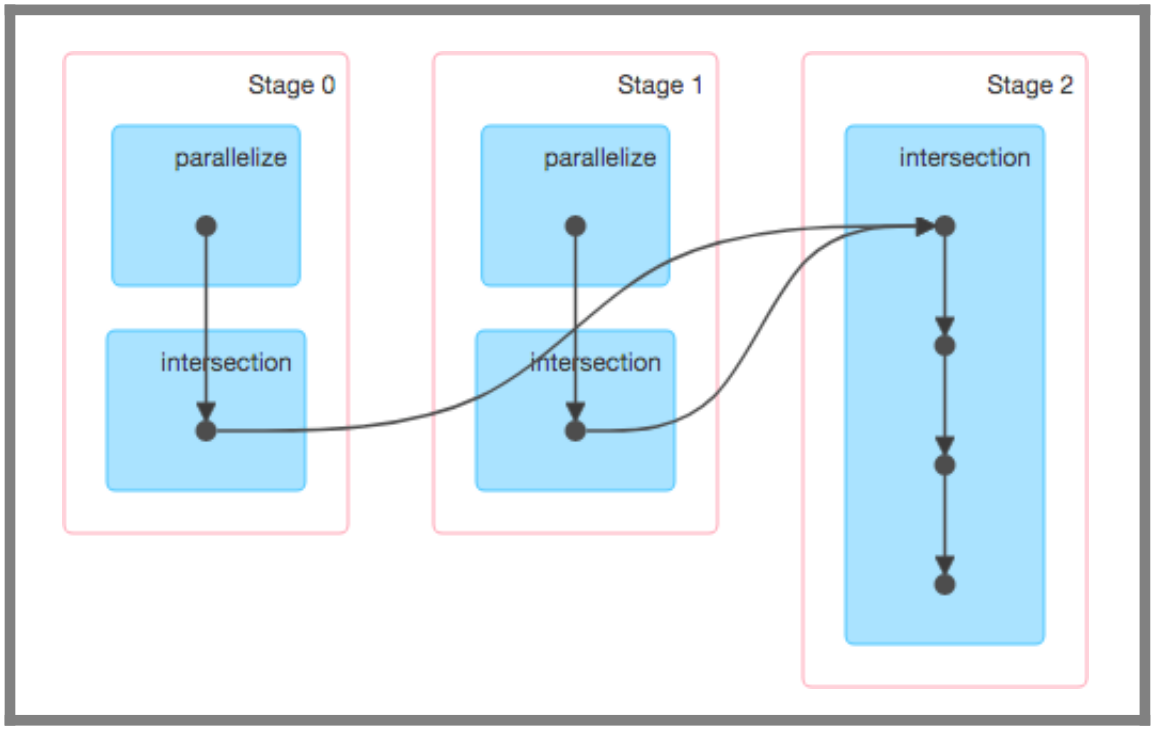
Tasks

- Create an RDD (u3) with the meetups it is attending (m1, m3, m5).
- Using your three RDDs, find the recommendations for u1 based on the rules above.
    - Look at the diagram above, and put into words how you would compute the recommendations.
    - Once you have a clear idea of how to do it, perform the RDD operations to accomplish it in the Spark Shell.
    - You'll need to use all 3 RDDs in your transformation for this.
- Look at the job DAG in the UI after you've done this.

Discussions on What's Seen in the Spark Shell UI

Let's consider at some of the results you might have seen earlier in the UI.

Here's the DAG from finding the meetups in common (an intersection of the two RDDs).



What you're seeing is this.

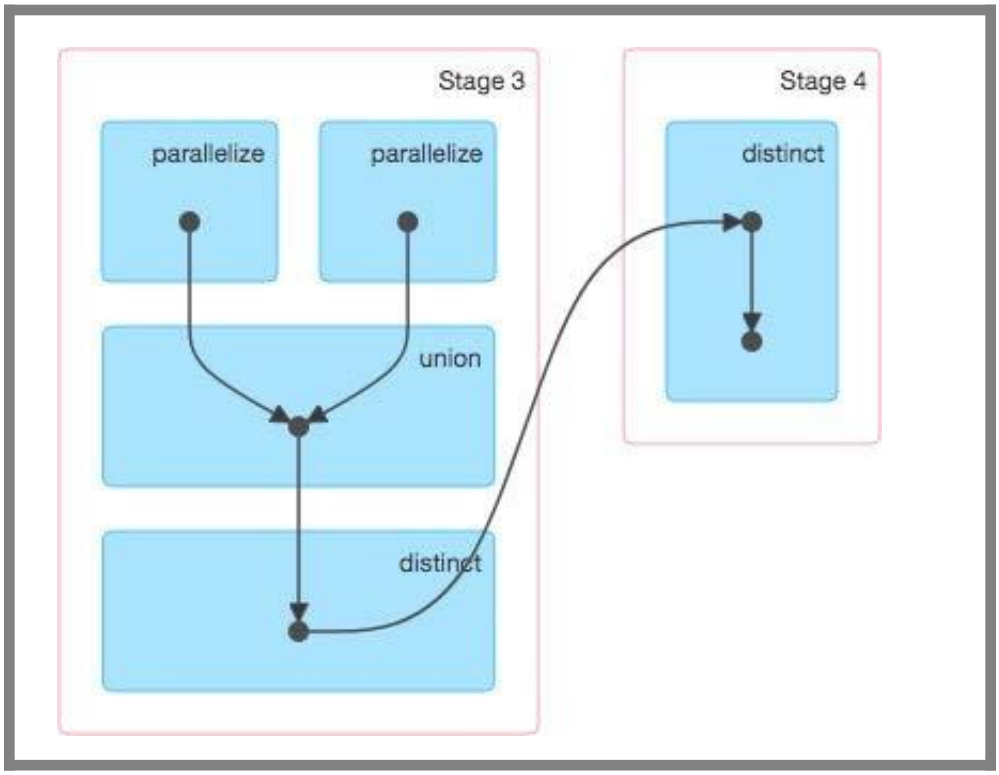
- Each shaded blue box represents a user operation.
- A dot in the box represents an RDD created in the corresponding operations.
- Operations are grouped by stages (In a stage, operations on partitions are pipelined in the same task).
- You can click on any stage, to drill down into it for more detail.

- Parallelization of each RDD occurs in one stage (e.g. on one node, with local data) Some of
- the intersection can happen on one stage (using whatever data is local) Some of the
- intersection happens in another stage
    - Because it can no longer be done with local data - it involves data distributed over the cluster.
    - Data is **shu"ed** for the intersection to be done (i.e. sent from one node to another).
    - Shu#ing is expensive - we'll talk more about this later.

Here are the details on the stages (from the **Completed Stages** section on the same page). It gives details on the data that is shu#ed.

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
2	collect at <console>:31 +details	2017/05/04 12:49:22	63 ms	8/8			364.0 B	
1	intersection at <console>:28 +details	2017/05/04 12:49:22	38 ms	8/8				156.0 B
0	intersection at <console>:28 +details	2017/05/04 12:49:22	0.2 s	8/8				208.0 B

Here's another DAG - from finding the meetups attended by **either** user (a union of the two RDDs).



What you're seeing is this.

Parallelization of each RDD occurs in one stage (e.g. on one node, with local data)

-

    - And yes, Spark has to parallelize again, because default is not to cache an RDD - more on this later.
- The union happens in the same stage - It can all be done with local data
    - Part of the distinct has to be done in a separate stage - it also requires shu#ing data

---

Summary

You've done some more complex transformations, working with very simple data. You've also viewed how the transformation works in the Spark Shell UI, and can see that even a fairly simple transformation can result in a complex set of operations in the Spark runtime. In later labs, we'll work with other complex operations, as well as with large data sets, to dig deeper into Spark's capabilities.