MSDS 694
Distributed Computing

DIANE WOODBRIDGE, PH.D

Announcement

Assignments

- Individual Assignment 3 - **Dec 3rd**
- Group Assignment 3 - **Dec 14th**
- Group Peer Review - **Dec 18th**
  - **Note : Based on this, your Group Assignment grade will be re-weighted.**

Office Hour : Tuesday 9:15-10 am

Professionalism

- **Class Attendance**
  - No cellphones, social media, slack, texting during the class .
  - Please only use laptops for class-related purposes.

- **Assignment**
  - No late submissions are allowed.
  - Do not share any homework and exam files - All the codes/reports including the last 6 years will be tested by Moss (Measure Of Software Similarity) and Turnitin.
  - Make sure that your code runs in Python 3 and Pyspark 3.

Contents

Advanced Topics in Spark

- Persist in Memory/Disk
- Tuning Parallelism
- RDD Dependencies

Appendix

- Advanced Topics in Spark
  - Example - PageRank
  - Using Shared Variables
  - Cluster Configuration

Contents

Advanced Topics in Spark

- **Persist in Memory/Disk**
- Tuning Parallelism
- RDD Dependencies

Appendix

- Advanced Topics in Spark
  - Example - PageRank
  - Using Shared Variables
  - Cluster Configuration

Tuning Spark – Persist in Memory/Disk

Lightning-fast cluster computing

1. Speed: Run programs up to 100x faster than Hadoop MapReduce in memory, or 10x faster on disk.

- Apache Spark has an advanced DAG (Directed Acyclic Graph) execution engine that supports cyclic data flow and in-memory computing.

http://spark.apache.org/

Tuning Spark – Persist in Memory/Disk

RDDs are by default recomputed each time.

- However, if you want to reuse an RDD for multiple actions, you can ask Spark to store the content in memory/disk to enhance the speed for querying repeatedly.

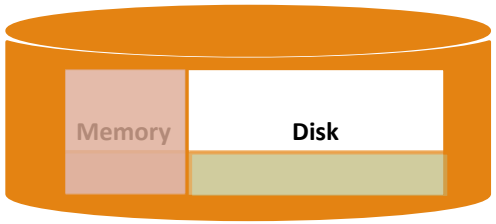
```
from pyspark.storagelevel import StorageLevel
rdd.persist(StorageLevel.persistency_level)
```

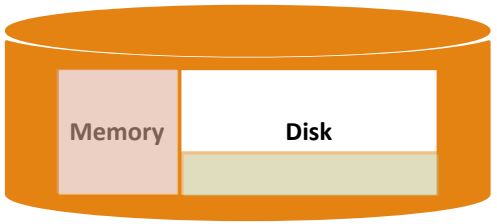
https://spark.apache.org/docs/latest/tuning.html

Tuning Spark – Persist in Memory/Disk

Persistency Level `rdd.persist(StorageLevel.persistency_level)`

Storage Level	Meaning
`MEMORY_ONLY` (Default)	Store RDDs in memory. If an RDD does not fit in memory, some partitions will not be cached and will be recomputed on the fly each time.
`DISK_ONLY`	Store RDD partitions only on disk.
`MEMORY_AND_DISK`	If an RDD does not fit in memory, store the partitions that don't fit on disk, and read them from there when they're needed.
`MEMORY_ONLY_2, MEMORY_AND_DISK_2, etc.`	Same as the levels above, but replicate each partition on two cluster nodes. (_N indicates the replication level)





https://spark.apache.org/docs/latest/rdd-programming-guide.html#rdd-persistence

Tuning Spark – Persist in Memory/Disk

Persist Methods

- **`.cache()`** is the same as calling `.persist(MEMORY_ONLY)`.
- **`.persist(Storage Level)`**
  - Many options (See the previous Table) - memory/disk and replication.
  - Defaults : `MEMORY_ONLY`
- **`.unpersist()`** lets you manually remove them from the storage.

https://spark.apache.org/docs/latest/tuning.html

Ex01

Apply power_map function to all the numbers in Data/numbers.txt using .map()

- After creating powered_num, does calling powered_num.collect() get faster?

```
class Power:
    def __init__(self, p):
        self.p = p
        time.sleep(2)

    def applyPower(self, x):
        return x**self.p

# map
def power_map(num):
    c = Power(5)
    return c.applyPower(num)
```

Ex01

Apply power_map function to all the numbers in Data/numbers.txt using .map()

- After creating powered_num, does calling powered_num.collect() get faster?

Without Persisting or Caching

```
start = time.time()
powered_num = numbers.map(power_map)
powered_num.collect()
print("first ", time.time() - start)

start = time.time()
powered_num.collect()
print("second ", time.time() - start)
```

```
first  42.91487002372742
second  42.058833837509155
```

Ex01

Apply power_map function to all the numbers in Data/numbers.txt using .map()

- How about after persisting it?

With Persisting/Caching

```
powered_num.cache()
start = time.time()
powered_num.collect()
print("first ", time.time() - start)  # trigger re-evaluation

start = time.time()
powered_num.collect()
print("second ", time.time() - start)  # doesn't trigger re-evaluation
```

```
first  42.07180404663086
second  0.0422189235687255 86
```

Ex01

Apply power_map function to all the numbers in Data/numbers.txt using .map()

- Change the persistency-level.

```
powered_num.persist(StorageLevel.MEMORY_AND_DISK)
```

```
---------------------------------------------------------------------------
Py4JJavaError                             Traceback (most recent call last)
<ipython-input-6-fea334107381> in <module>
----> 1 powered_num.persist(StorageLevel.MEMORY_AND_DISK)

~/anaconda3/envs/DistributedComputing/lib/python3.7/site-packages/pyspark/rdd.py in persist(self, storageLevel)
    244         self.is_cached = True
    245         javaStorageLevel = self.ctx._getJavaStorageLevel(storageLevel)
--> 246         self._jrdd.persist(javaStorageLevel)
    247         return self
    248

~/anaconda3/envs/DistributedComputing/lib/python3.7/site-packages/py4j/java_gateway.py in __call__(self, *args)
   1255         answer = self.gateway_client.send_command(command)
   1256         return_value = get_return_value(
-> 1257             answer, self.gateway_client, self.target_id, self.name)
   1258
   1259         for temp_arg in temp_args:

~/anaconda3/envs/DistributedComputing/lib/python3.7/site-packages/py4j/protocol.py in get_return_value(answer, gatewa
y_client, target_id, name)
    326                 raise Py4JJavaError(
    327                     "An error occurred while calling {0}{1}{2}.\n".
--> 328                     format(target_id, ".", name), value)
    329             else:
    330                 raise Py4JError(

Py4JJavaError: An error occurred while calling o26.persist.
: java.lang.UnsupportedOperationException: Cannot change storage level of an RDD after it was already assigned a leve
l
```

In order to fix the error, what do you need to do?

- power_num.persist(StorageLevel.MEMORY_ONLY)
- power_num.unpersist()
- power_num.persist(StorageLevel.DISK_ONLY)
- None of the above

In order to fix the error, what do you need to do?

power_num.persist (StorageLevel.MEMORY_ONLY)

power_num.unpersist()

power_num.persist (StorageLevel.DISK_ONLY)

None of the above

In order to fix the error, what do you need to do?

power_num.persist(StorageLevel.MEMORY_ONLY)

power_num.unpersist()

power_num.persist(StorageLevel.DISK_ONLY)

None of the above

Ex01

Apply power_map function to all the numbers in Data/numbers.txt using .map()

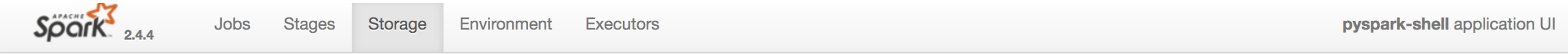
- Change the persistency-level.

```
powered_num.persist(StorageLevel.DISK_ONLY)
start = time.time()
powered_num.collect()
print("first ", time.time() - start)  # trigger reevaluation

start = time.time()
powered_num.collect()
print("second ", time.time() - start)  # doesn't trigger reevaluation
```

Tuning Spark – Persist in Memory/Disk

Persisted RDDs in the Spark Application UI.



Storage

▾ RDDs

ID	RDD Name	Storage Level	Cached Partitions	Fraction Cached	Size in Memory	Size on Disk
3	PythonRDD	Memory Serialized 1x Replicated	5	100%	658.0 B	0.0 B

Tuning Spark – Persist in Memory/Disk

Tips

- If your RDDs fit comfortably with MEMORY_ONLY, leave them that way. This is the most CPU-efficient option, allowing operations on the RDDs to run as fast as possible. Iterative algorithms are often good candidates for caching.
- Don’t spill to disk unless the functions take long time to run or they use a large amount of the data. Otherwise, recomputing a partition may be as fast as reading it from disk.
- Use the replicated storage levels if you want fast fault recovery (e.g. if using Spark to serve requests from a web application). All the storage levels provide full fault tolerance by recomputing lost data, but the replicated ones let you continue running tasks on the RDD without waiting to recompute a lost partition.

Contents

Advanced Topics in Spark

- Persist in Memory/Disk
- **Tuning Parallelism**
- RDD Dependencies

Appendix

- Advanced Topics in Spark
  - Example - PageRank
  - Using Shared Variables
  - Cluster Configuration

2022 Interview Question

Hi Diane, how is everything going? I have just been asked a spark related interview question that I am not sure what the answer is. The question is how rdd join with each other, and how can we improve that? (especially between two large datasets or between a small dataset and a large dataset)

Tuning Spark - The Level of Parallelism

Partitions

- For parallel collections, users can specify the min number of partitions to cut the RDD into.
- Definition
  - `rdd = sc.parallelize(data, numSlices) or textFile(file_name, minPartitions)`
  - `sc.parallelize(data).transformation(…,numPartitions)`
    - Operations taking the number of partitions.
      - `sortBy(), sortByKey(), groupBy(), reduceByKey(), join(), leftOuterJoin(), rightOuterJoin(), partitionBy(), combineByKey(), aggregateByKey(), foldByKey(), groupByKey(), subtractByKey(), subtract(), repartition(), coalesce(), etc.`

https://spark.apache.org/docs/latest/api/java/org/apache/spark/Partitioner.html
http://spark.apache.org/docs/latest/api/python/_modules/pyspark/context.html#SparkContext.parallelizel
http://spark.apache.org/docs/latest/api/python/pyspark.html#pyspark.SparkContext.textFile

Tuning Spark - The Level of Parallelism

Partitions

Check :

- `rdd.getNumPartitions()`
  - `getNumPartitions()` : Returns the number of partitions in RDD
- `rdd.glom().collect()`
  - `glom()`: Return an RDD created by coalescing all elements within each partition into a list.

https://spark.apache.org/docs/latest/api/python/pyspark.html

Ex02

For given partition_ct, solve the following problems.

1. Create an RDD which has partition_ct as the number of partitions in a format of (zip, supervisor_id) order by zip.
   - How the data is organized? Do same keys locate in the same partition?

```
[(94102, 8),
 (94102, 6),
 (94102, 3),
 (94102, 5),
 (94103, 8),
 (94103, 9),
 (94103, 10),
 (94103, 6),
 (94103, 3),
 (94103, 5),
 (94104, 6),
 (94104, 3),
 (94105, 6),
 (94105, 3),
 (94107, 10),
 (94107, 6),
 (94108, 6),
 (94108, 3),
 (94109, 2),
 (94109, 6),
 (94109, 3),
 (94109, 5)],
[(94110, 8),
 (94110, 11),
 (94110, 9),
 (94110, 10),
 (94111, 6),
 (94111, 3),
 (94112, 7),
 (94112, 8),
 (94112, 11),
 (94112, 9),
 (94112, 10),
```

Ex02

For given partition_ct, solve the following problems.

1. Create an RDD which has partition_ct as the number of partitions in a format of (zip, supervisor_id) order by zip.
   - How the data is organized? Do same keys locate in the same partition?

```
sorted_zip_supervisor_id = supervisor_sf.map(lambda x : x.split('\t'))\
                                        .map(lambda x : (int(x[0]), int(x[1])))\
                                        .sortByKey(numPartitions=partition_ct)
```

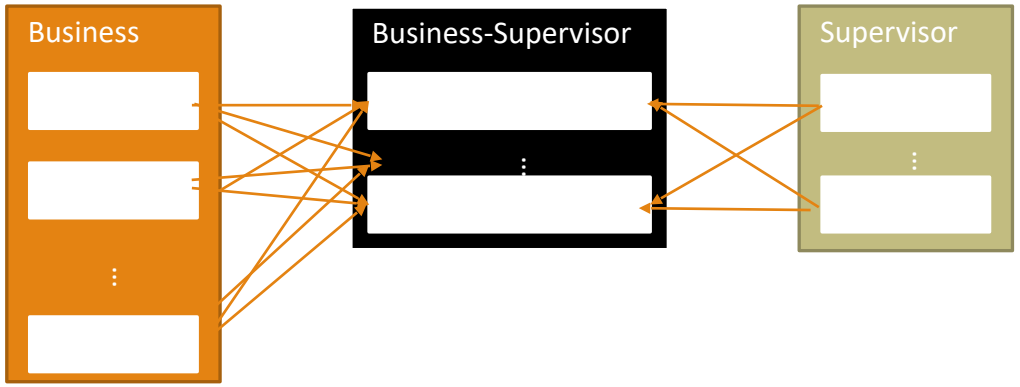
How about joining with business?

```
[(94102, 8),
 (94102, 6),
 (94102, 3),
 (94102, 5),
 (94103, 8),
 (94103, 9),
 (94103, 10),
 (94103, 6),
 (94103, 3),
 (94103, 5),
 (94104, 6),
 (94104, 3),
 (94105, 6),
 (94105, 3),
 (94107, 10),
 (94107, 6),
 (94108, 6),
 (94108, 3),
 (94109, 2),
 (94109, 6),
 (94109, 3),
 (94109, 5)],
[(94110, 8),
 (94110, 11),
 (94110, 9),
 (94110, 10),
 (94111, 6),
 (94111, 3),
 (94112, 7),
 (94112, 8),
 (94112, 11),
 (94112, 9),
 (94112, 10),
```

Tuning Spark - The Level of Parallelism

Problem

- Sending data back and forth between executors on parallelized distributed system causes network traffic.
  - Better to place data that can minimize shuffling and improve performance.
- Workload might not be evenly distributed in some partitions.
  - May cause efficiency or memory issues.



https://spark.apache.org/docs/latest/api/java/org/apache/spark/Partitioner.html

```
lines = sc.textFile("filtered_registered_business_sf.csv")
zip_business = lines.map(lambda x: (x.split("\t")[0],x[1:])).persist()
```

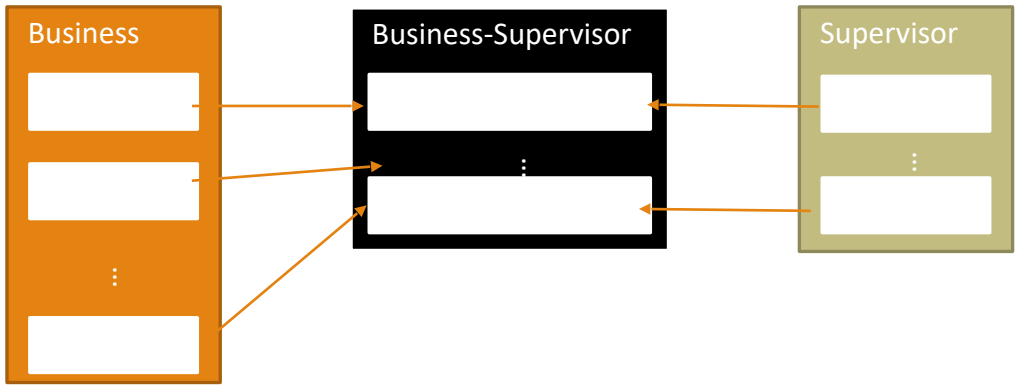
Tuning Spark - The Level of Parallelism

Solution

Partitioner

Which operations would benefit from .partitionBy()?

- RDDs : **`.partitionBy(partition_size, partitionFunc)`**
  - Defines how the elements in an RDD or key-value pair RDD are partitioned.
    - You can manage data commonly accessible together on the same node.
    - Organize data for minimizing network traffic/communication to improve performance.
    - Maps each key to a partition ID, from 0 to 'numPartitions - 1'.
  - Types : HashPartitioner (default), RangePartitioner, Custom Partitioner.



https://spark.apache.org/docs/latest/api/java/org/apache/spark/Partitioner.html

Tuning Spark - The Level of Parallelism

Solution

Partitioner

- RDDs: .**`partitionBy(partition_size, partitionFunc)`**
  - Defines how the elements in an RDD or key-value pair RDD are partitioned.
  - Types : HashPartitioner, RangePartitioner, Custom Partitioner.
    - HashPartitioner : partitioner using a hash value of a key
      - .partitionBy(N) uses HashPartitioner by default.
    - RangePartitioner : partitioner partitioning sorted RDDs into roughly equal ranges.
    - CustomPartitioner : User-defined partitioner.

ex)

```
def custom_partitioner(key):
    return hash(key + 10)

pair_rdd.partitonBy(N, custom_partitioner)
```

Ex02

For given partition_ct, solve the following problems.

2. Create an RDD which has partition_ct as the number of partitions in a format of (zip, [name, street, city, zip]) **partitioned by zip** (hash).
   - How the data is organized? Do same keys locate in the same partition?

3. Join RDD from 1) and 2) efficiently for any RDDs in 1).
   - Assume that the joined RDDs will be repeatedly used.

```
[(94124,
  ['Stephens Institute Inc', '2225 Jerrold Ave', 'San Francisco', 'CA']),
 (94108, ['Stephens Institute Inc', '540 Powell St', 'San Francisco', 'CA']),
 (94108, ['Stephens Institute Inc', '740 Taylor St', 'San Francisco', 'CA']),
 (94108,
  ['"Fugazi Travel Agency, Inc."',
   '170 Grant Ave 4th Fl',
   'San Francisco',
   'CA']),
 (94132, ['Agid Hyman D', '452 Gellert Dr', 'San Francisco', 'CA']),
 (94108, ['Felton Donald & Nancy', '445 Stockton St', 'San Francisco', 'CA']),
 (94108, ['Felton Donald & Nancy', '445 Stockton St', 'San Francisco', 'CA']),
 (94124, ['Anresco Inc', '1370 Van Dyke Ave', 'San Francisco', 'CA']),
 (94124, ['Anresco Inc', '1370 Van Dyke Ave', 'San Francisco', 'CA']),
 (94108,
  ['Anthonys Shoe Service Inc', '340 Kearny St', 'San Francisco', 'CA']),
 (94112, ['Antonchuk Richard A', '1117 Ocean Ave', 'San Francisco', 'CA']),
 (94112, ['Antonchuk Richard A', '1125 Ocean Ave', 'San Francisco', 'CA']),
 (94124, ['Arnold & Egan Mfg Co', '1515 Griffith St', 'San Francisco', 'CA']),
```

For a repeatedely used RDD, using .cache()/.persist() can improve its efficiency.

True	**A**
False	**B**

For a repeatedely used RDD, using .cache()/.persist() can improve its efficiency.

True	**A**
False	**B**

For a repeatedely used RDD, using .cache()/.persist() can improve its efficiency.

True	**A**
False	**B**

Ex02

partitionBy() + cache() 😀

For given partition_ct, solve the following problems.

2. Create an RDD which has partition_ct as the number of partitions in a format of (zip, [name, street, city, zip]) partitioned by zip (hash).
   - How the data is organized? Do same keys locate in the same partition?

```
business_sf.map(lambda x : x.split('\t'))\
           .filter(lambda x: len(x[0]) > 0)\
           .map(lambda x : (int(x[0]), x[1:]))\
           .partitionBy(numPartitions=partition_ct)
```

3. Join RDD from 1) and 2) efficiently for any RDDs in 1).
   - Assume that the joined RDDs will be repeatedly used.

partitionBy() : Spark know that it is hash-partitioned.
persist() : Without persisting an RDD after partitioning, it will cause subsequent uses of the RDD to repeat the partitioning of the data.
→Much Faster

```
sorted_zip_supervisor_id = sorted_zip_supervisor_id.cache()
partitioned_zip_business = partitioned_zip_business.cache()
joined_supervisor_business = sorted_zip_supervisor_id.leftOuterJoin(partitioned_zip_business)
```

Ex02

For given partition_ct, solve the following problems.

4. For the range of supervisor ids, create RDD of (supervisor_id, [business name, street, city, state]) within the particion_ct based on the supervisor_id ranges.
   - Ex. If partition_ct is 3, and supervisor ids are [1, 2, 3, 4, 5, 6, 7, 8, 9] => 1st partition should have pairs with supervisor ids of [1,2,3], 2nd partition should have pairs with supervisor ids of [4,5,6], 3rd partition should have pairs with supervisor ids of [7,8,9]
5. Store pairs of (supervisor_id, [business name, street, city, state]) within 2 partitions, based on whether the supervisor_id is even or odd.

Ex02

Extra Tip : To avoid unexpected errors, you may want to consider to use broadcast variable for min_id, max_id, and particion_ct (See Appendix)

For given partition_ct, solve the following problems.

4. For the range of supervisor ids, create RDD of (supervisor_id, [business name, street, city, state]) within the particion_ct based on the supervisor_id ranges.

```
def custom_range_partition(key):
    min_id = supervisor_ids[0]
    max_id = supervisor_ids[-1]
    partition_size = (max_id - min_id + 1)/partition_ct
    return int((key - min_id)/partition_size)
```

```
supervisor_business_custom_range = supervisor_business.partitionBy(numPartitions=partition_ct,
                                                                    partitionFunc=custom_range_partition)
```

5. Store pairs of (supervisor_id, [business name, street, city, state]) within 2 partitions, based on whether the supervisor_id is even or odd.

```
def odd_even_partitioner(key):
    return key%2
```

```
supervisor_business_odd_even = supervisor_business.partitionBy(2, odd_even_partitioner)
```

Tuning Spark - The Level of Parallelism

Operations and Partitioning

1. **Operations benefiting from partitioning**
   - Operations involving shuffling data by key across the network.
     - Ex. join(), leftOuterJoin(), rightOuterJoin(), groupByKey(), reduceByKey(), combineByKey(), lookUp(), etc.
2. **Operations returns RDDs with known partitioning information.**
   - sortByKey() : range-partition
   - groupByKey() : hash-partition
3. **Operations forget the parent's partitioning information.**
   - map() : because it can theoretically modify the key of each record
4. **Most operations not mentioned here inherit from their parents.**

Tuning Spark - The Level of Parallelism

Operations and Partitioning

- **Operations repartitioning RDDs.**
  - Change the partitioning to distribute the workload more efficiently or avoid memory problems.
    1. `.repartition(numPartitions: Int)`
       - Shuffle data across the network to create a new set of partitions. ➔Expensive.
    2. `.coalesce(numPartitions: Int, shuffle = false)`
       - **Optimized** version of repartition() – avoid data movement and **reduce** the number of RDD partitions.
       - Match the locality as much as possible, but try to balance partitions across the machines.

Ex03

Compare `.coalesce()` and `.repartition()`.

- Can the number of partitions smaller than its parent's number of partitions?
- Can the number of partitions bigger than its parent's number of partitions?
- Which one shuffles data less?

Ex03

Compare `.coalesce()` and `.repartition()`.

- Can the number of partitions smaller than its parent's number of partitions?
- Can the number of partitions bigger than its parent's number of partitions?
- Which one shuffles data less?

```
nums = sc.parallelize([1,1,3,3,11,13,14],6)
nums.glom().collect()
```

```
[[1], [1], [3], [3], [11], [13, 14]]
```

```
#also works for pair RDDs.
print(nums.repartition(3).glom().collect())
print(nums.coalesce(3).glom().collect())
```

```
[[], [1, 3, 13, 14], [1, 3, 11]]
[[1, 1], [3, 3], [11, 13, 14]]
```

```
print(nums.glom().collect())
print(nums.repartition(10).glom().collect())
print(nums.coalesce(10).glom().collect())
print(nums.coalesce(10, True).glom().collect())
```

```
[[1], [1], [3], [3], [11], [13, 14]]
[[], [1], [3, 3], [1], [11], [], [], [], [], [13, 14]]
[[1], [1], [3], [3], [11], [13, 14]]
[[], [1], [3, 3], [1], [11], [], [], [], [], [13, 14]]
```

Contents

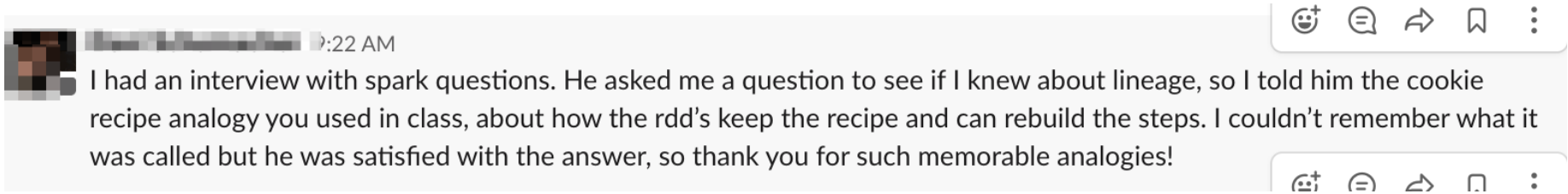
Advanced Topics in Spark

- Persist in Memory/Disk
- Tuning Parallelism
- **RDD Dependencies**

Appendix

- Advanced Topics in Spark
  - Example - PageRank
  - Using Shared Variables
  - Cluster Configuration

2018 Interview Questions



RDD Dependencies (RDD Lineage)

A dependency between an old and a new RDD is created, every time a transformation is performed on an RDD.

- Spark's execution model is based on directed acyclic graphs (DAGs), where nodes are RDDs and edges are dependencies.
  - The new RDD depend on the old RDD.
- RDD Resilience - As Spark records the linage of each RDD, any RDDs can be reconstructed to the state it was at the time of the failure using RDD lineage.

RDD Dependencies (RDD Lineage)

Spark stages and tasks

- Every job is divided into stages based on the points where shuffles occur.
  - For each stage, tasks are created and sent to the executors.
  - After all tasks of a particular stage complete, the driver creates tasks for the next stage and sends them to the executors.

RDD Dependency Types

1. Narrow – When no data shuffle between partitions is required.
2. Wide - When it requires shuffle when joining RDDs.

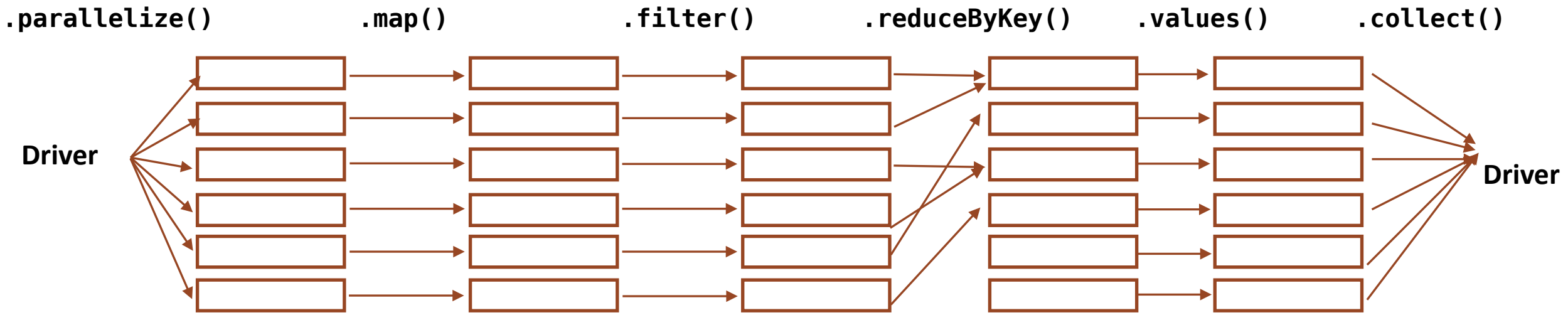
Ex04

What are the types of dependencies in the following code (Each line)??

```
nums = sc.parallelize([1, 1, 3, 3, 11, 13, 14], 6)
mapped_nums = nums.map(lambda x: (x, x - 1))
filtered_nums = mapped_nums.filter(lambda x: x[1] > 0)
reduced_nums = filtered_nums.reduceByKey(lambda x, y: x + y)
values = reduced_nums.values()
values.collect()
```

Ex04

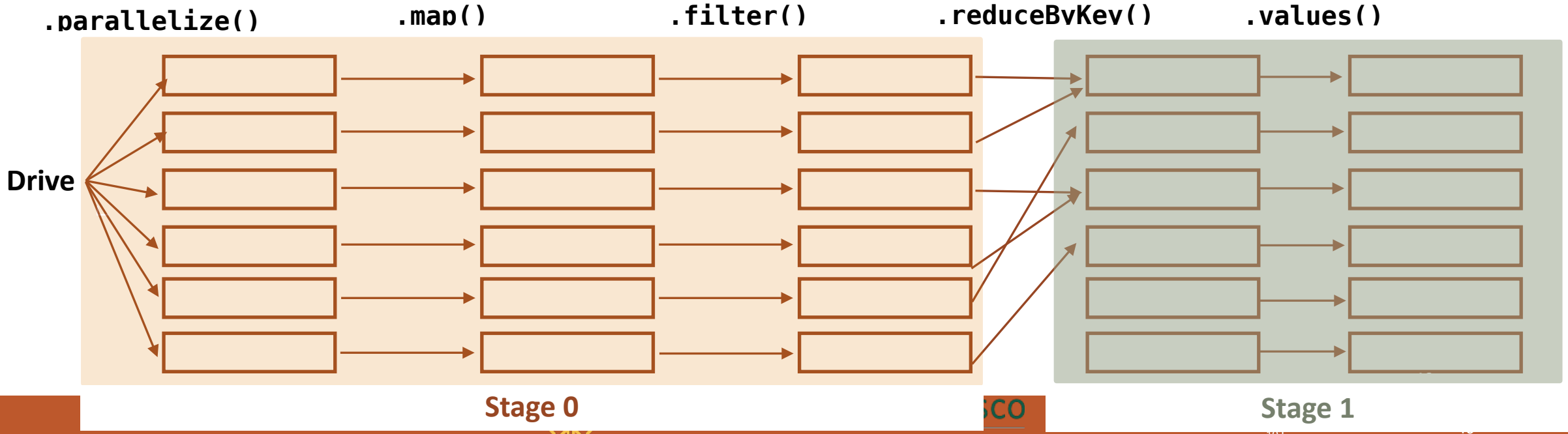
```
nums = sc.parallelize([1, 1, 3, 3, 11, 13, 14], 6)
mapped_nums = nums.map(lambda x: (x, x - 1))
filtered_nums = mapped_nums.filter(lambda x: x[1] > 0)
reduced_nums = filtered_nums.reduceByKey(lambda x, y: x + y)
values = reduced_nums.values()
values.collect()
```



RDD Dependencies (RDD Lineage)

Spark stages and tasks in Example 4.

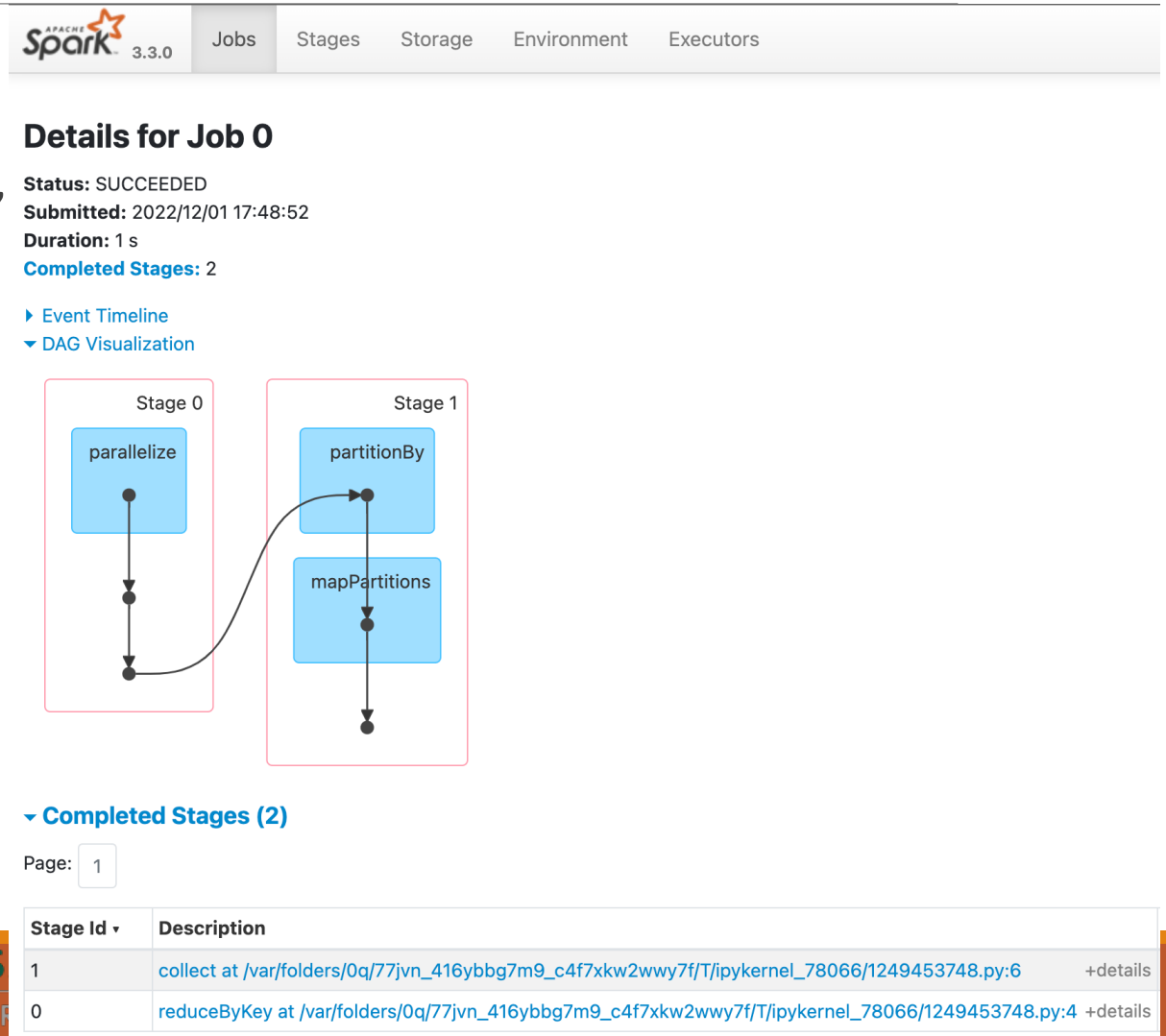
- Stage 0 encompasses transformations that result in a shuffle: parallelize , map , filter and reduceByKey .
- During Stage 1, each partition receives data and the execution is continued.



RDD Dependencies (RDD Lineage)

Spark stages and tasks in Example 4.

- Stage 0 encompasses transformations that result in a shuffle: parallelize , map , filter and reduceByKey .
- During Stage 1, each partition receives data and the execution is continued.



RDD Dependencies (RDD Lineage)

`.toDebugString()`

- Shows a textual representation of RDD dependencies.
- The RDDs in the output appears in reverse order.
- Useful in trying to minimize the number of shuffles.
  - The numbers in parentheses show the number of partitions of the corresponding RDD.
- Every time you see a ShuffleRDD in the lineage chain, you can be sure that a shuffle will be performed at that point.

```
print(values.toDebugString().decode("utf-8"))
```

```
(6) PythonRDD[30] at RDD at PythonRDD.scala:53 []
 |  MapPartitionsRDD[20] at mapPartitions at PythonRDD.scala:145 []
 |  ShuffledRDD[19] at partitionBy at NativeMethodAccessorImpl.java:0
[]
 +-(6) PairwiseRDD[18] at reduceByKey at /var/folders/0q/77jvn_416ybbg
7m9_c4f7xkw2wwy7f/T/ipykernel_78066/2663297619.py:1 []
    |  PythonRDD[17] at reduceByKey at /var/folders/0q/77jvn_416ybbg7m
9_c4f7xkw2wwy7f/T/ipykernel_78066/2663297619.py:1 []
    |  ParallelCollectionRDD[6] at readRDDFromFile at PythonRDD.scala:
274 []
```

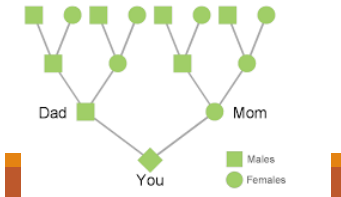
RDD Dependencies (RDD Lineage)

`.toDebugString()`

- Shows a textual representation of RDD dependencies.
- The RDDs in the output appears in reverse order.
- Useful in trying to minimize the number of shuffles.
  - The numbers in parentheses show the number of partitions of the corresponding RDD.
- Every time you see a ShuffleRDD in the lineage chain, you can be sure that a shuffle will be performed at that point.

```
print(values.toDebugString().decode("utf-8"))
```

```
(6) PythonRDD[30] at RDD at PythonRDD.scala:53 []
 |  MapPartitionsRDD[20] at mapPartitions at PythonRDD.scala:145 []
 |  ShuffledRDD[19] at partitionBy at NativeMethodAccessorImpl.java:0
[]
 +-(6) PairwiseRDD[18] at reduceByKey at /var/folders/0q/77jvn_416ybbg
7m9_c4f7xkw2wwy7f/T/ipykernel_78066/2663297619.py:1 []
    |  PythonRDD[17] at reduceByKey at /var/folders/0q/77jvn_416ybbg7m
9_c4f7xkw2wwy7f/T/ipykernel_78066/2663297619.py:1 []
    |  ParallelCollectionRDD[6] at readRDDFromFile at PythonRDD.scala:
274 []
```



Contents

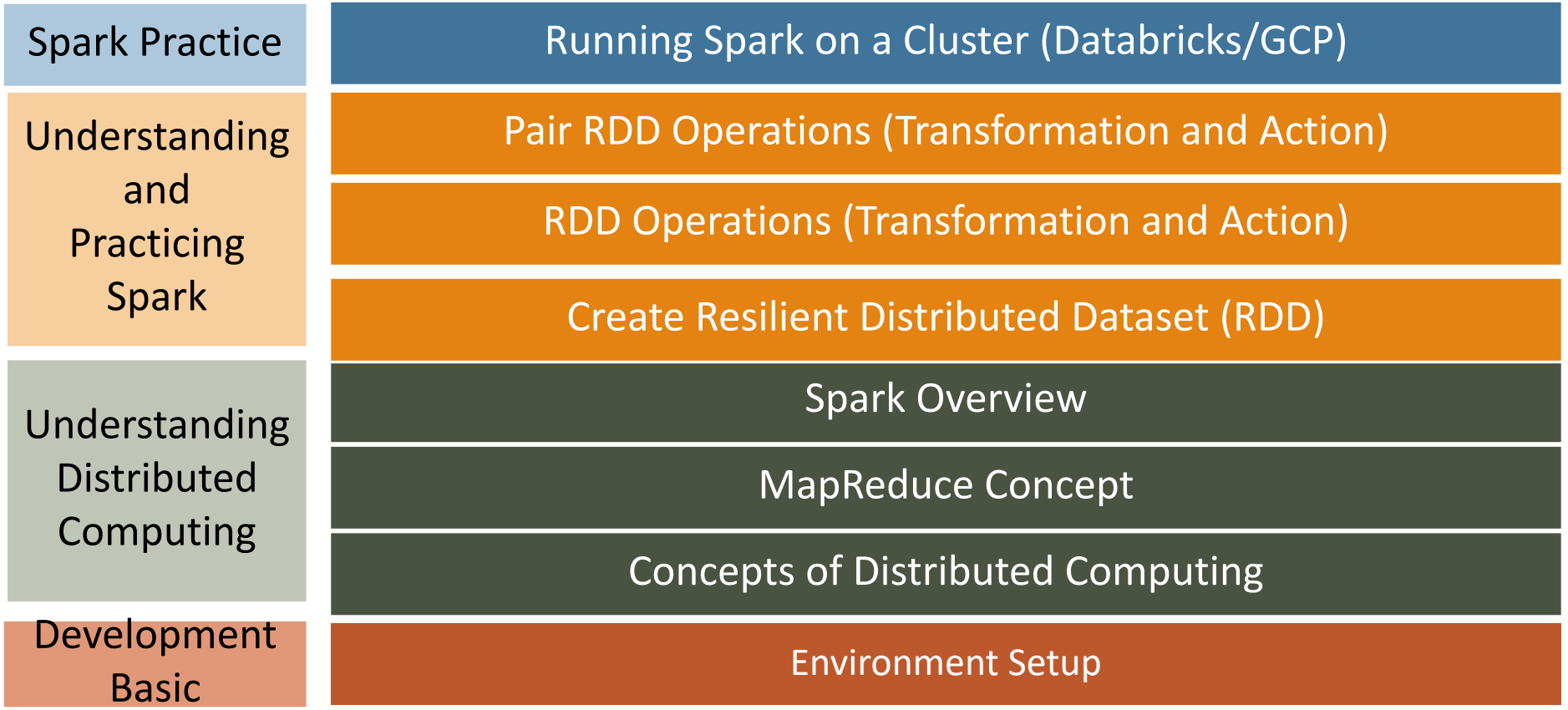
Advanced Topics in Spark

- Persist in Memory/Disk
- Tuning Parallelism
- RDD Dependencies

Appendix

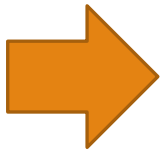
- Advanced Topics in Spark
  - Example - PageRank
  - Using Shared Variables
  - Cluster Configuration

About Distributed Computing (MSDS694)



About Distributed Computing(MSDS694)

Distributed Computing (MSDS 694)
Distributed Computing Concept Basic Spark Operations Running Spark on Clusters Advanced Topics in Spark



Distributed Data Systems (MSDS 697)
NoSQL MongoDB Spark SQL Spark ML Apache AirFlow

Plus, some practicum companies require Spark.

Teaching Effectiveness Survey

PLEASE COMPLETE FOR EVERY CLASS.

IT IS IMPORTANT FOR OUR TENURE AND PROMOTION AND IMPROVING THE QUALITY OF EDUCATION AT MSDS.

Reference

1. Spark Online Documentation, http://spark.apache.org/docs/latest/
2. Zecevic, Petar, et al. Spark in Action, Manning, 2016.
3. Databricks on Google Cloud, https://docs.gcp.databricks.com/

Appendix : Shared Variable, Cluster Configuration

Contents

Advanced Topics in Spark

- Example - PageRank
- Using Shared Variables

Cluster Configuration

Contents

Advanced Topics in Spark

- **Example - PageRank**
- Using Shared Variables

Cluster Configuration

PageRank

Determine a measure of importance (a “rank”) to each document based on how many documents have links to it.

Applications : Rank web pages, influential users in a social network, etc.

(Also could be implemented using GraphX.)

Algorithm

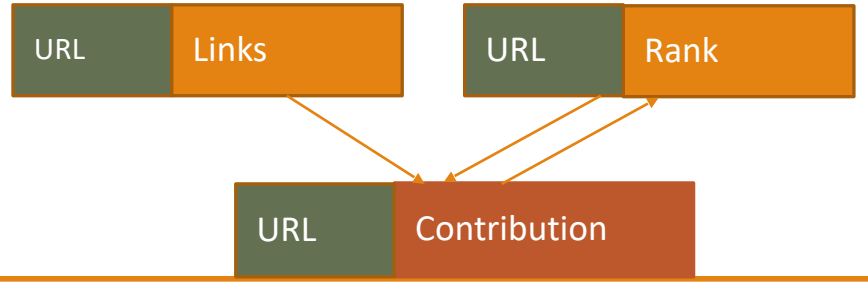
1. Read each URL and its links information.

2. Initialize each page’s rank to 1.

3. Iterate :

contribution = rank/numNeighbors

rank = 0.15 + 0.18 * sum(contribution_received).



https://github.com/apache/spark/blob/master/examples/src/main/python/pagerank.py
https://www.safaribooksonline.com/library/view/learning-spark/9781449359034/ch04.html#idp10672624

ex - PageRank

Write a page rank algorithms where data = [(1,[2,3,4]), (2,[1,3]), (3,[4])] where the format is (URL, [LIST OF URLS]).

ex - PageRank

Algorithm

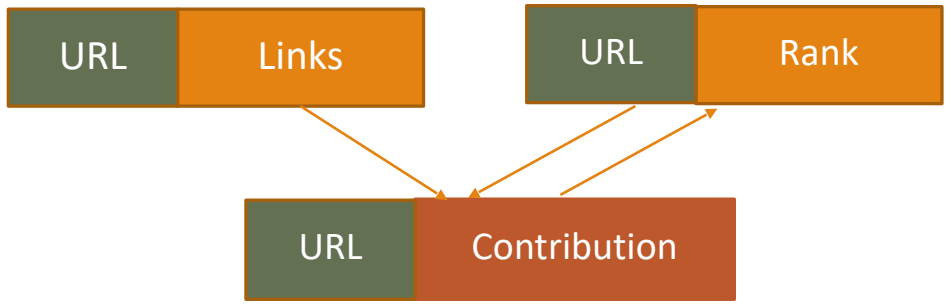
1. Read each URL and its links information.

2. Initialize each page's rank to 1.

3. Iterate :

contribution = rank/numNeighbors

rank = 0.15 + 0.18 * sum(contribution_received).



Since links is a static dataset, we partition it at the start with partitionBy(), so that it does not need to be shuffled across the network.
We call cache() (or persist()) on links to keep it in RAM across iterations.

https://github.com/apache/spark/blob/master/examples/src/main/python/pagerank.py
https://www.safaribooksonline.com/library/view/learning-spark/9781449359034/ch04.html#idp10672624

ex - PageRank

Algorithm

1. Read each URL and its links information.

2. Initialize each page's rank to 1.

3. Iterate :

contribution = rank/numNeighbors

rank = 0.15 + 0.18 * sum(contribution_received).

```
data =  [(1,[2,3,4]), (2,[1,3]), (3,[4])] #simplified version : could be URL, LIST OF URLS

links = sc.parallelize(data).partitionBy(2).cache()

ranks = links.map(lambda x: (x[0], 1.0)) #init each rank to 1

#compute each url's contribution
def computeContribs(urls, rank):
    for url in urls:
        yield (url, rank / len(urls))

it_num = 10 #in practice it runs about 10 iterations
for i in range (it_num):
    contributions = links.join(ranks).flatMap(lambda x : computeContribs(x[1][0],x[1][1]))
    ranks = contributions.reduceByKey(lambda x,y : x+y).mapValues(lambda x : x*0.85+0.15)
ranks.collect()
```

```
[(1, 0.2430319557083253),
 (2, 0.21887340110327186),
 (3, 0.3119053568115972),
 (4, 0.4840575158297381)]
```

partitionBy().persist() or cache() improves efficiency.

Since links is a static dataset, we partition it at the start with partitionBy(), so that it does not need to be shuffled across the network.
We call cache() (or persist()) on links to keep it in RAM across iterations.

https://github.com/apache/spark/blob/master/examples/src/main/python/pagerank.py
https://www.safaribooksonline.com/library/view/learning-spark/9781449359034/ch04.html#idp10672624

Contents

Advanced Topics in Spark

- Example - PageRank
- **Using Shared Variables**

Cluster Configuration

Shared Variables

Normally, when a function passed to a Spark operation is executed, it works on separate copies of all the variables used in the function. These variables are copied to each partition, and updates to the variables are not propagated back to the driver program.

Solution : Shared variables

- Help maintain a global state or share data across tasks and partitions.
  1. **Accumulator** : Aggregate information from executor nodes to the driver node.
  2. **Broadcast variable** : Efficiently distribute large read-only values to executor nodes.

http://spark.apache.org/docs/latest/rdd-programming-guide.html#shared-variables

Shared Variables

Using shared variables to communicate with Spark executors.

Shared variables

1. Accumulator
   - Aggregate information from executor nodes to the driver node.
   - Is shared across executors that you can only add to.
     - Useful to implement global sums and counters.
   - Can be accessed by driver node, not from an executor node.
     - The driver can call it using **`.value`**
   - The executor can add to the accumulator with .**`add`**(val) method or **`+`**. (it is **write-only**.)
   - Create using `var_name = sc.accumulator(inital_value)`

http://spark.apache.org/docs/latest/rdd-programming-guide.html#shared-variables

Shared Variables

Using shared variables to communicate with Spark executors.

Shared variables

1. Accumulator
   - If accumulators are used in transformation, the results may be erroneous.
     - Can cause accumulator values to be counted more than once if tasks or job stages are re-executed.
     - Accumulator updates are not guaranteed to be executed when made within a lazy transformation.
     - Should use accumulators within actions such as foreach().
   - `foreach(f)`
     - `Action`
     - Applies a function, f to all elements

https://spark.apache.org/docs/latest/rdd-programming-guide.html#accumulators
https://spark.apache.org/docs/latest/api/python/pyspark.html?highlight=foreach

Ex05

Define an accumulator variable and initialize the value to be 0.

Generate values between 1 and 10,000,000 using .parallelize().

For each value, increment accumulator variable.

http://spark.apache.org/docs/latest/programming-guide.html#shared-variables

Ex05

Within Transformation

```
list_2 = list.map(lambda x : accum.add(1))
```

```
accum.value
```

```
0
```

```
list_2.first()
```

```
accum.value
```

```
1
```

Within Action

```
accum = sc.accumulator(0)
```

```
list.foreach(lambda x : accum.add(1))
```

```
accum.value
```

```
10000000
```

Within Transformation

- Can cause accumulator values to be counted more than once if tasks or job stages are re-executed.
- Accumulator updates are not guaranteed to be executed when made within a lazy transformation.

http://spark.apache.org/docs/latest/programming-guide.html#shared-variables

Ex05

```
list.foreach(lambda x: accum.value)
```

```
7, in processPartition
    f(x)
  File "/Users/dwoodbridge/anaconda3/envs/DistributedComputing/lib/python3.7/site-packages/pyspark/util.py", line 9
9, in wrapper
    return f(*args, **kwargs)
  File "<ipython-input-11-72c55adb883b>", line 1, in <lambda>
  File "/Users/dwoodbridge/anaconda3/envs/DistributedComputing/
rk.zip/pyspark/accumulators.py", line 153, in value
    raise Exception("Accumulator.value cannot be accessed inside tasks")
Exception: Accumulator.value cannot be accessed inside tasks

        at org.apache.spark.api.python.BasePythonRunner$ReaderIterator.handlePythonException(PythonRunner.scala:45
6)
        at org.apache.spark.api.python.PythonRunner$$anon$1.read(PythonRunner.scala:592)
        at org.apache.spark.api.python.PythonRunner$$anon$1.read(PythonRunner.scala:575)
        at org.apache.spark.api.python.BasePythonRunner$ReaderIterator.hasNext(PythonRunner.scala:410)
        at org.apache.spark.InterruptibleIterator.hasNext(InterruptibleIterator.scala:37)
        at scala.collection.Iterator$class.foreach(Iterator.scala:891)
        at org.apache.spark.InterruptibleIterator.foreach(InterruptibleIterator.scala:28)
        at scala.collection.generic.Growable$class.$plus$plus$eq(Growable.scala:59)
```

Exception: Accumulator.value cannot be accessed inside tasks

Exception occurs when you tried to access accumulator values in executor nodes.

http://spark.apache.org/docs/latest/programming-guide.html#shared-variables

Ex06

Data

bike-share/status_million.csv includes station information and include 4 fields with a format of "***station_id, num_bikes_available, num_docks_available, timestamp***"

Ex06

Count how many status data are collected where station_id is ’10’.

Compare an output of .count() from 1) an accumulator value incremented within a transformation and 2) an accumulator value incremented within an action.

Ex06

Accumulator used in a transformation vs action.

```
def filter_station(x):
    if (x[1][0]== '10'):
        accumulator_in_transformation.add(1)
        return x

filtered_station_status = status.filter(filter_station)
filtered_station_status.foreach(lambda x: accumulator_in_action.add(1))
```

```
print(filtered_station_status.count())
print(accumulator_in_transformation.value)
print(accumulator_in_action.value)
```

```
523623
1570869
523623
```

http://spark.apache.org/docs/latest/programming-guide.html#shared-variables

Shared Variables

Using shared variables to communicate with Spark executors.

Shared variables

2. Broadcast variable : Read-only object set by the driver.
   - Efficiently distribute large read-only values such as lookup tables to executor nodes.
   - The value is sent to each node only once (not per task).
   - Create a broadcast variable using `var_name = sc.broadcast(value).`
   - Access the value with `.value`.
   - Should **`.unpersist()`** for removing a broadcast variable from memory on all workers. - It still stays on the driver node.

http://spark.apache.org/docs/latest/programming-guide.html#shared-variables

Shared Variables

Using shared variables to communicate with Spark executors.

Shared variables

2. Broadcast variable : Read-only object set by the driver.
   - Advantages
     - Replicate data once per worker (not once per task).
     - Are serialized objects (can be read efficiently.).

http://spark.apache.org/docs/latest/programming-guide.html#shared-variables

Ex07

Generate a broadcast variable and access, modify and unpersist it.

Ex07

Generate a broadcast variable and access, modify and unpersist it.

```
list = sc.parallelize(range(10,100))
```

```
broadcastVar = sc.broadcast([1, 2, 3])
```

```
type(broadcastVar)
```

```
pyspark.broadcast.Broadcast
```

```
broadcastVar.value
```

```
[1, 2, 3]
```

```
add_broadcastVar = list.map(lambda x : x + broadcastVar.value[0])
```

```
add_broadcastVar.collect()
```

Ex07

Generate a broadcast variable and access, modify and unpersist it.

```
broadcastVar = broadcastVar + 1
broadcastVar.value
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-7-cfbfce962bf2> in <module>()
----> 1 broadcastVar = broadcastVar + 1
      2 broadcastVar.value

TypeError: unsupported operand type(s) for +: 'Broadcast' and 'int'
```

```
broadcastVar.unpersist() #unpersist method removes from executor nodes. It still stays on the driver node, so it can be
broadcastVar.value
```

```
[3, 2, 1]
```

Contents

Advanced Topics in Spark

- Example - PageRank
- Using Shared Variables

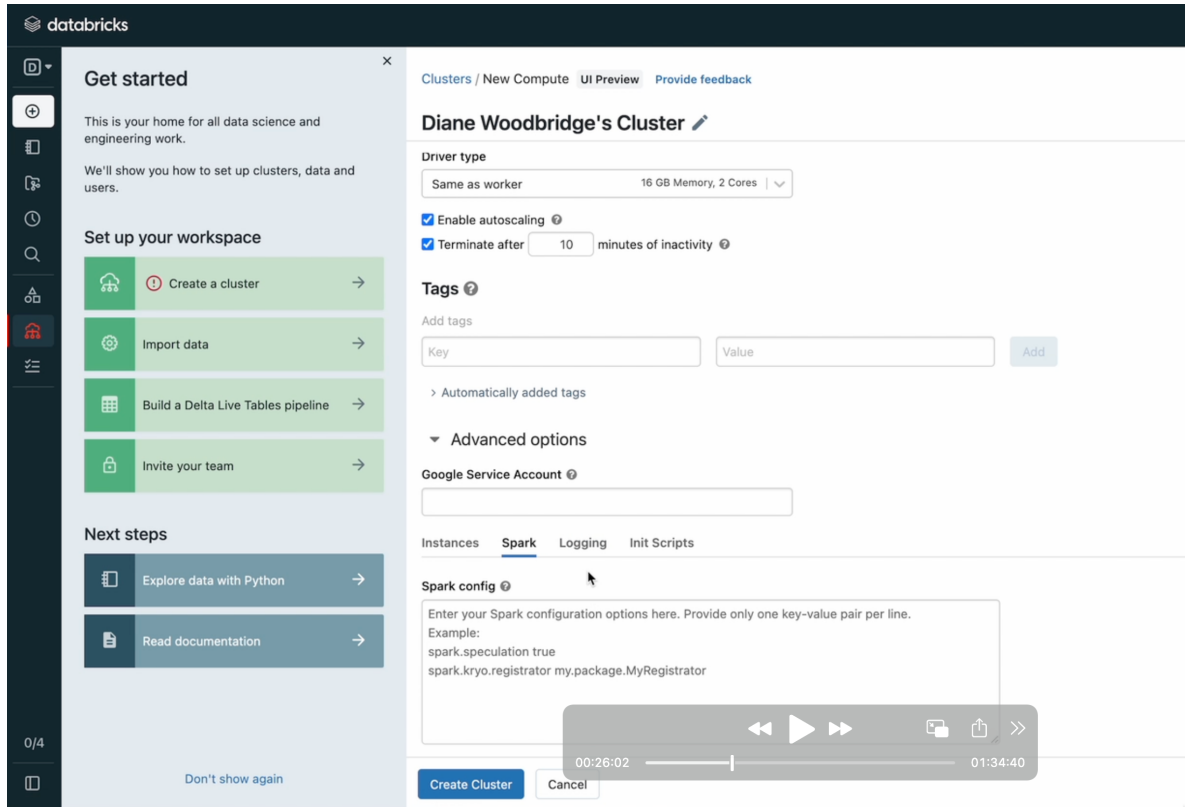
Cluster Configuration

Databricks Cluster Configuration

Under your workspace, you can create clusters and configure them.

To configure a cluster

- Choose **Advanced Options**
- Configure the default values : Specify variables and value under “Spark”



What can I configure?

Property Name	Default	Meaning
spark.app.name	(none)	The name of your application.
spark.driver.maxResultSize	1g	Limit of total size of serialized results of all partitions for each Spark action (e.g. collect) in bytes. Setting a proper limit can protect the driver from out-of-memory errors.
spark.executor.memory	1g	Amount of memory to use per executor process, in the same format as JVM memory strings with a size unit suffix ("k", "m", "g" or "t") (e.g. 512m, 2g).

https://spark.apache.org/docs/latest/configuration.html

What can I configure?

Property Name	Default	Meaning
spark.executor.memoryOverhead	executorMemory * 0.10, with minimum of 384	The amount of off-heap memory to be allocated per executor, in MiB unless otherwise specified. This is memory that accounts for things like VM overheads, interned strings, other native overheads, etc. This tends to grow with the executor size (typically 6-10%). This option is currently supported on YARN and Kubernetes.
spark.executor.cores	1 in YARN mode, all the available cores on the worker in standalone and Mesos coarse-grained modes.	The number of cores to use on each executor. In standalone and Mesos coarse-grained modes, for more detail, see this description.

https://spark.apache.org/docs/latest/configuration.html