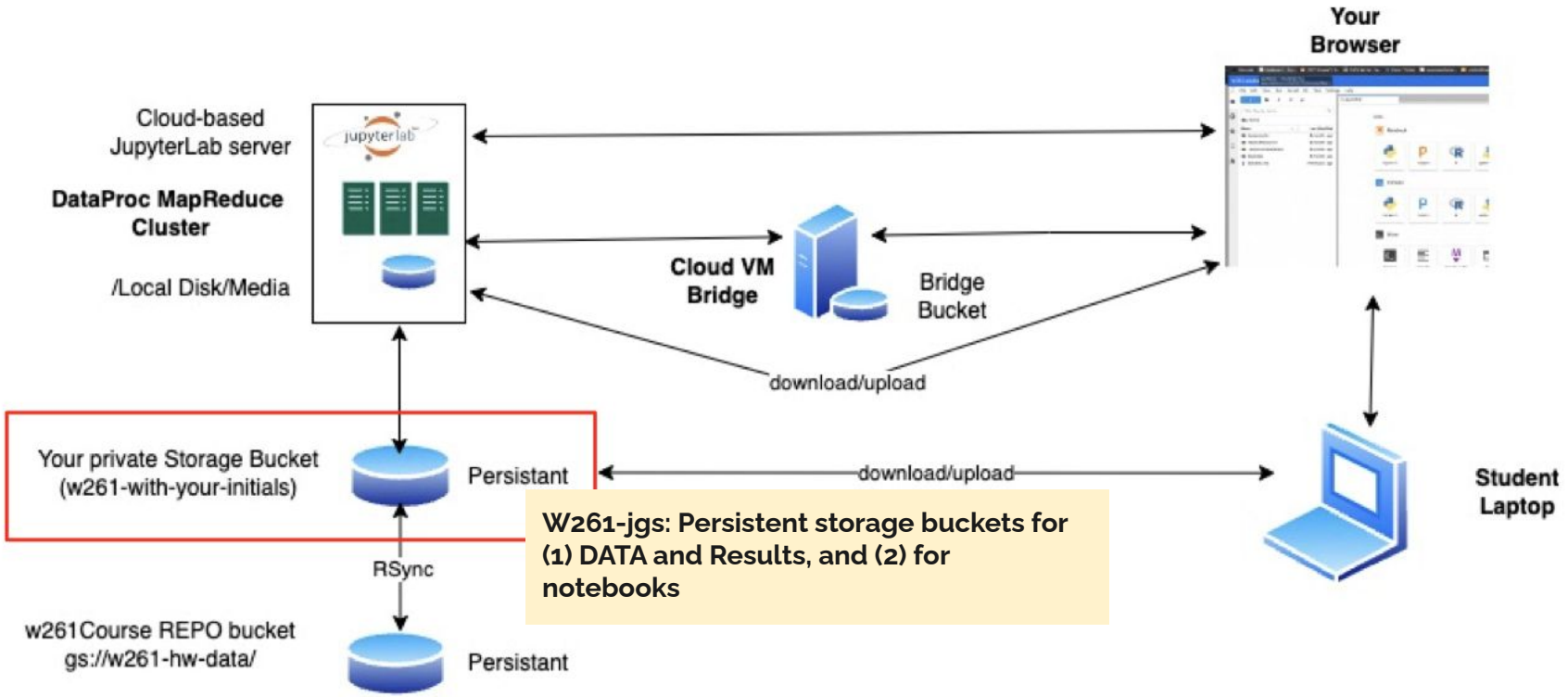
W261 - Week 5
Introduction to SPARK, part 2

UC Berkeley - MIDS
2023

Table of Contents

The big migration to cloud buckets and Spark!

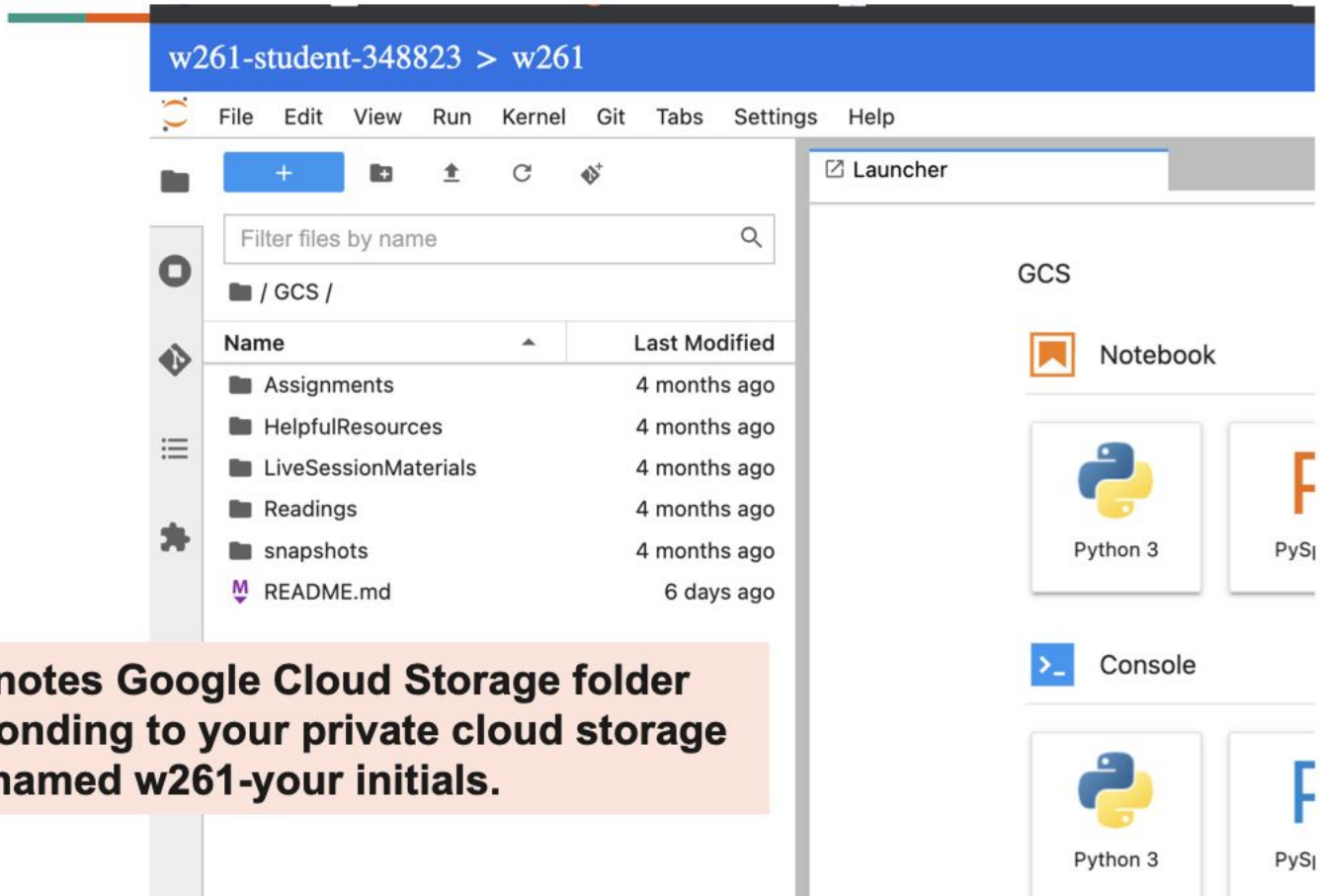
Cloud-based JupterLab (backed by a DataProc Cluster)



W261-jgs: Persistent storage buckets for (1) DATA and Results, and (2) for notebooks

Orchestration shell script cmd: gsutil cat gs://w261-hw-data/w261_env.sh | bash -euo pipefail

Next steps: Working directly with cloud buckets and Spark Jobs



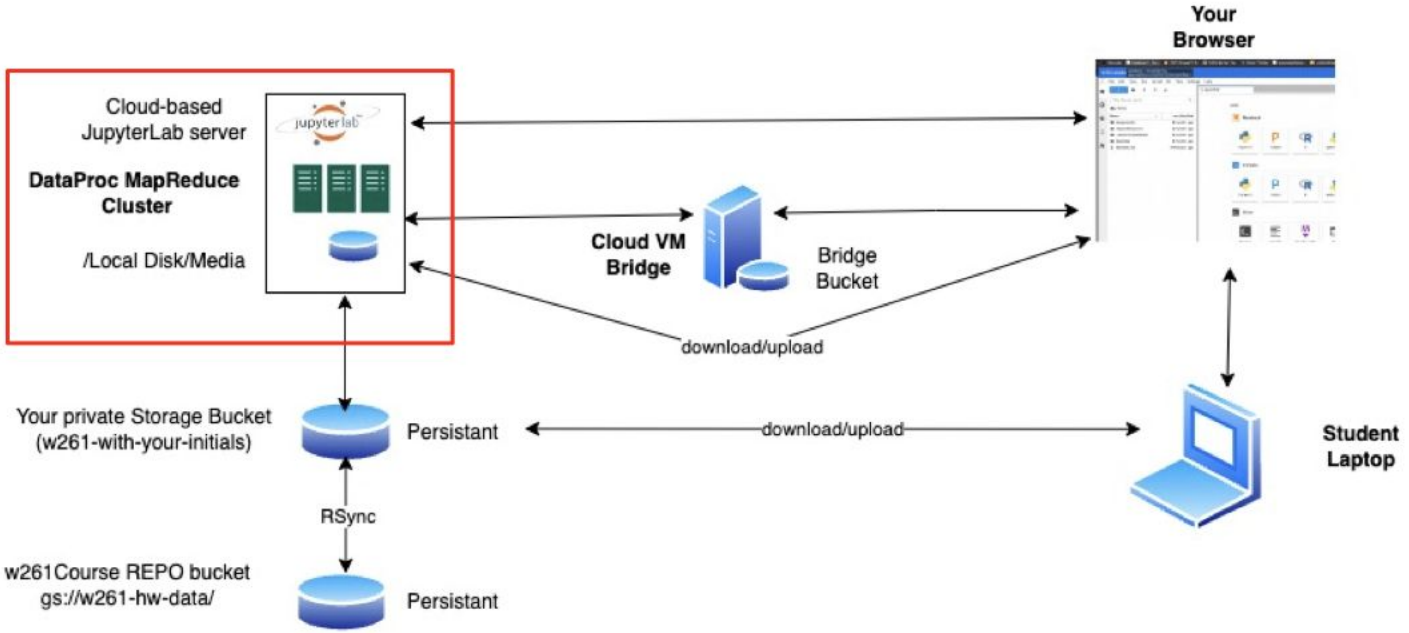
GCS denotes Google Cloud Storage folder corresponding to your private cloud storage bucket named w261-your initials.

JupterLab PaaS at $0.20 per hour

Dataproc is a fully managed and highly scalable service for running Hadoop, Apache Spark, Apache Flink, Presto, and 30+ open source tools and frameworks.

Setup a hadoop/Spark (aka DataProc) cluster on Google cloud with a **single node** (4 CPUs)
The cluster costs $0.20 per hour and has 100 Gig of local Disk

~$35 per week
~ $300 for 9 weeks



Click here to access CloudShell/Bridge: https://console.cloud.google.com/

Orchestration shell script cmd: gsutil cat gs://w261-hw-data/w261_env.sh | bash -euo pipefail

Data Bucket and Staging bucket updates

- **Data and HW output vs notebook Locations: private data bucket versus dataproc staging bucket**
  - **When you create a DataProc cluster, HDFS is used as the default filesystem. In this course, we override this behavior by setting the defaultFS as a Cloud Storage bucket (your private bucket DATA_BUCKET).** The benefits of using Cloud Storage buckets are that your job results get to persist beyond the lifetime of the cluster (and btw latency on these cloud buckets is super low).

- ~~**STAGING_BUCKET: notebook locations (please IGNORE as no longer used)**~~
  - ~~Associated with each DataProc cluster is a persistant storage bucket that we refer to as the DataProc Staging Bucket. You will be using this staging bucket to store notebooks and other files associated with your HW assignments, and live sessions. The location of the staging bucket is made available via os.getenv("STAGING_BUCKET"). Since this bucket is persistent, we will no longer need to snapshot your student workspaces.~~
  - ~~`os.getenv("STAGING_BUCKET")`~~

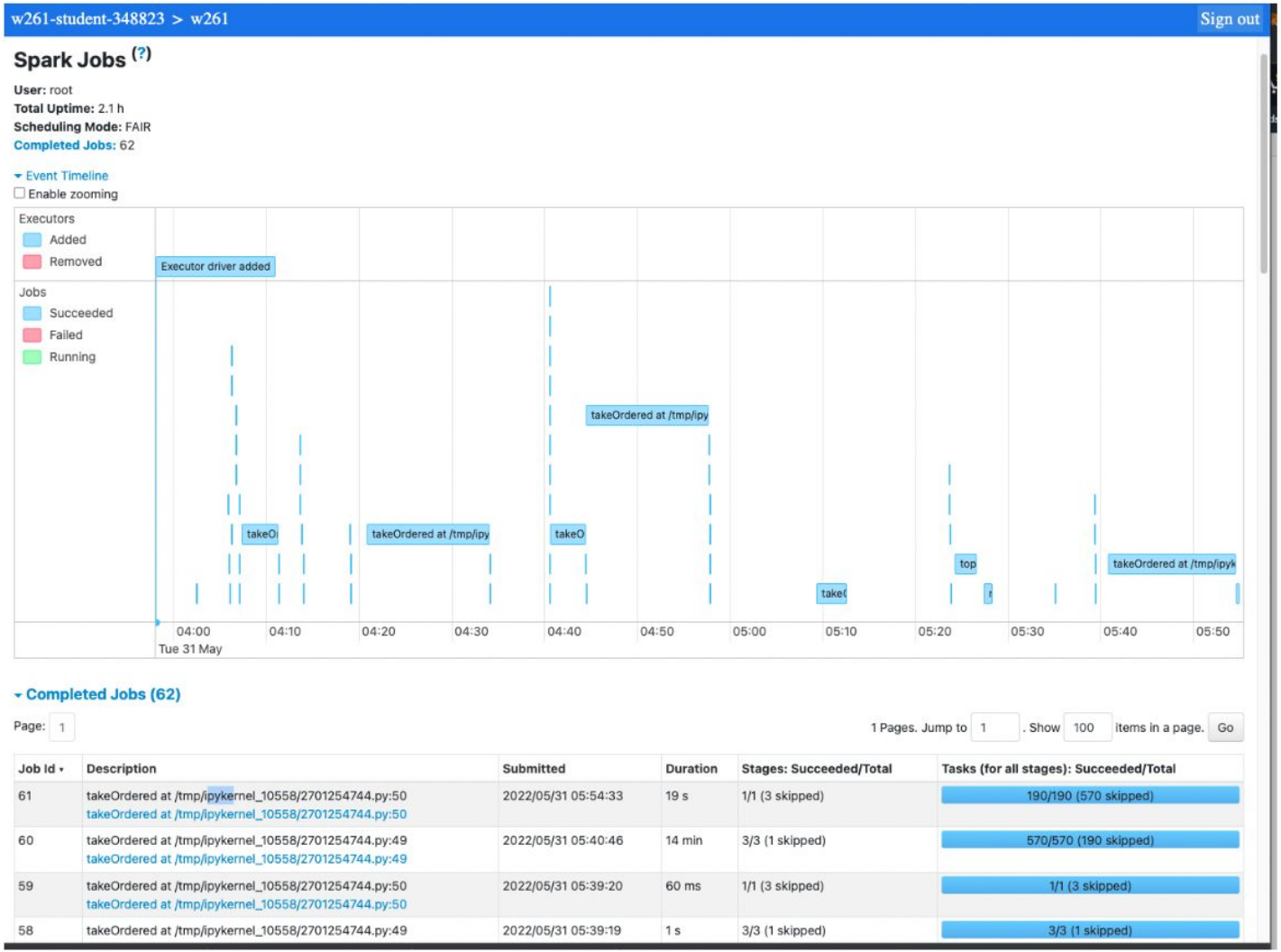
- **DATA_BUCKET: Data and HW output**
  - In this HW, you use a cloud bucket (and folders on them), known as DATA_BUCKET, as input and output for the Spark Apps that you will develop as part of your submission.
  - The datasets for this homework are preloaded into your private Data Bucket on Google Cloud. Recall that you created a private data bucket during the first week of semester. You may have called it w261-. Jimi's bucket name is w261-jgs. To facilitate ease of access, we have set up location variables for the course-related data buckets. Your private data bucket can be accessed via:
    - `os.getenv('DATA_BUCKET','')`
  - This DATA_BUCKET will be used for hosting the input data for this assignment and also to store output from your HW Assignment apps.

Save files to Data Bucket (HW3_FOLDER)

Test file: **systems_test.txt**

[ ]:
```
systems_test= """it was the best of	1	1	1
age of wisdom it was	1	1	1
best of times it was	1	1	1
it was the age of	2	1	1
it was the worst of	1	1	1
of times it was the	2	1	1
of wisdom it was the	1	1	1
the age of wisdom it	1	1	1
the best of times it	1	1	1
the worst of times it	1	1	1
times it was the age	1	1	1
times it was the worst	1	1	1
was the age of wisdom	1	1	1
was the best of times	1	1	1
was the age of foolishness	1	1	1
was the worst of times	1	1	1
wisdom it was the age	1	1	1
worst of times it was	1	1	1"""


systems_test_loc = f'{HW3_FOLDER}/systems_test.txt'

!echo "{systems_test}" | gsutil cp - {systems_test_loc}
```

Accessing Spark UI on DataProc and what does a Spark consist of? (Jobs, stages, and tasks)


w261-student-348823 > w261
Sign out
Spark Jobs (?)
User: root
Total Uptime: 2.1 h
Scheduling Mode: FAIR
Completed Jobs: 62
Event Timeline
Enable zooming
Executors
Added
Removed
Executor driver added
Jobs
Succeeded
Failed
Running
takeOrdered at /tmp/ipy
takeO
takeOrdered at /tmp/ipy
takeO
top
takeOrdered at /tmp/ipyk
take(
04:00
Tue 31 May
04:10
04:20
04:30
04:40
04:50
05:00
05:10
05:20
05:30
05:40
05:50
Completed Jobs (62)
Page: 1
1 Pages. Jump to 1 . Show 100 items in a page. Go
Job Id
Description
Submitted
Duration
Stages: Succeeded/Total
Tasks (for all stages): Succeeded/Total
61
takeOrdered at /tmp/ipykernel_10558/2701254744.py:50
takeOrdered at /tmp/ipykernel_10558/2701254744.py:50
2022/05/31 05:54:33
19 s
1/1 (3 skipped)
190/190 (570 skipped)
60
takeOrdered at /tmp/ipykernel_10558/2701254744.py:49
takeOrdered at /tmp/ipykernel_10558/2701254744.py:49
2022/05/31 05:40:46
14 min
3/3 (1 skipped)
570/570 (190 skipped)
59
takeOrdered at /tmp/ipykernel_10558/2701254744.py:50
takeOrdered at /tmp/ipykernel_10558/2701254744.py:50
2022/05/31 05:39:20
60 ms
1/1 (3 skipped)
1/1 (3 skipped)
58
takeOrdered at /tmp/ipykernel_10558/2701254744.py:49
2022/05/31 05:39:19
1 s
3/3 (1 skipped)
3/3 (1 skipped)

Spark Jobs (?)

User: root
Total Uptime: 25.9 h
Scheduling Mode: FAIR
Completed Jobs: 106
Failed Jobs: 4

▸ Event Timeline

▾ Completed Jobs (106)

Page: 1 2 >

2 Pages. Jump to 1 . Show 100 items in a page. G

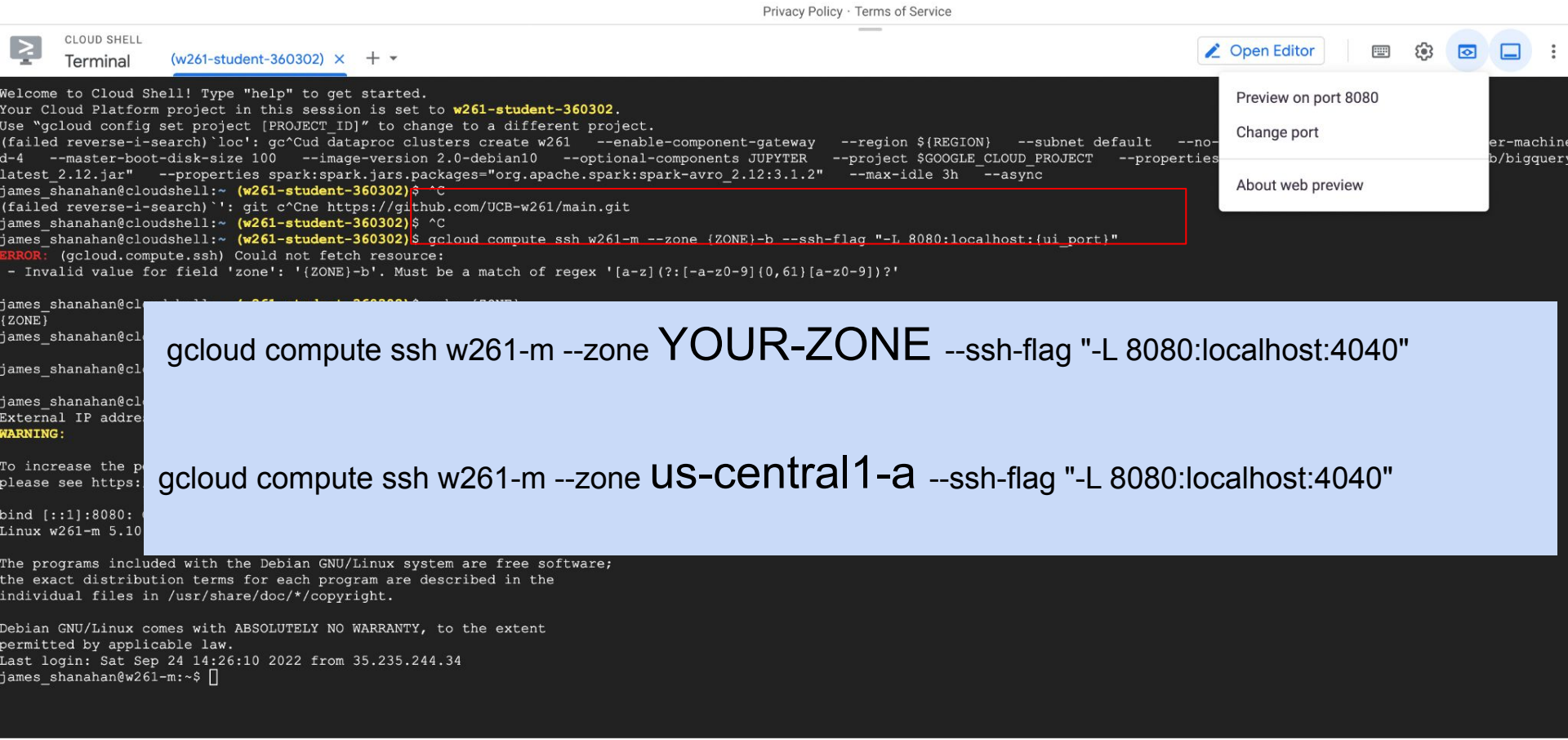
Job Id ▾	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
108	collect at /tmp/ipykernel_11411/2158800487.py:2 collect at /tmp/ipykernel_11411/2158800487.py:2	2022/09/24 18:38:15	0.2 s	2/2	4/4
106	collect at /tmp/ipykernel_11411/1175973261.py:5 collect at /tmp/ipykernel_11411/1175973261.py:5	2022/09/24 18:32:25	0.2 s	2/2	4/4
105	collect at /tmp/ipykernel_11411/1175973261.py:3 collect at /tmp/ipykernel_11411/1175973261.py:3	2022/09/24 18:32:25	0.1 s	2/2	4/4
104	collect at <timed exec>:45 collect at <timed exec>:45	2022/09/24 18:27:58	78 ms	2/2	4/4
103	collect at <timed exec>:45 collect at <timed exec>:45	2022/09/24 18:27:58	74 ms	2/2	4/4
102	takeSample at <timed exec>:38 takeSample at <timed exec>:38	2022/09/24 18:27:58	10 ms	1/1	2/2
101	takeSample at <timed exec>:38 takeSample at <timed exec>:38	2022/09/24 18:27:58	10 ms	1/1	2/2
100	collect at <timed exec>:45 collect at <timed exec>:45	2022/09/24 18:27:58	78 ms	2/2	4/4
99	collect at <timed exec>:45	2022/09/24 18:27:58	75 ms	2/2	4/4

Step A: Start Spark

```
]:
```

```
#Step A: Start Spark by running the following cell

from pyspark.sql import SparkSession

try:
    spark
except NameError:
    print('starting Spark')
    app_name = 'wk4_demo'
    master = "local[*]"
    spark = SparkSession\
            .builder\
            .appName(app_name)\
            .master(master)\
            .getOrCreate()
sc = spark.sparkContext
```

Step B:



gcloud compute ssh w261-m --zone YOUR-ZONE --ssh-flag "-L 8080:localhost:4040"

gcloud compute ssh w261-m --zone us-central1-a --ssh-flag "-L 8080:localhost:4040"

CLOUD SHELL

Terminal (w261-student-360302) ×

```
NOTE: In order to be able to see the SPARK UI, you need to have a running Notebook with an active Spark Context.
james_shanahan@cloudshell:~ (w261-student-360302)$ gcloud compute ssh w261-m --zone us-central1-a --ssh-flag "-L 8080:localhost:4040"
WARNING: The private SSH key file for gcloud does not exist.
WARNING: The public SSH key file for gcloud does not exist.
WARNING: You do not have an SSH key for gcloud.
WARNING: SSH keygen will be executed to generate a key.
This tool needs to create the directory [/home/james_shanahan/.ssh] before being able to generate SSH keys.

Do you want to continue (Y/n)?  Y

Generating public/private rsa key pair.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/james_shanahan/.ssh/google_compute_engine
Your public key has been saved in /home/james_shanahan/.ssh/google_compute_engine.pub
The key fingerprint is:
SHA256:KAqu8LHO/Q7uHiDIb4sR4DTY2JfL0JYNy76xYPwYZL8 james_shanahan@cs-798348507309-default
The key's randomart image is:
+---[RSA 3072]----+
|    .            |
|.+ o *           |
|+o* O .          |
|*+.B . .         |
|++* B . S        |
|.+oB *           |
|.o+oE            |
|oo+=.o           |
|.o=+=oo          |
+----[SHA256]-----+
External IP address was not found; defaulting to using IAP tunneling.
Updating project ssh metadata...working..Updated [https://www.googleapis.com/compute/v1/projects/w261-student-360302].
Updating project ssh metadata...done.
Waiting for SSH key to propagate.
WARNING:

To increase the performance of the tunnel, consider installing NumPy. For instructions,
please see https://cloud.google.com/iap/docs/using-tcp-forwarding#increasing_the_tcp_upload_bandwidth

Warning: Permanently added 'compute.7499392942176644213' (ECDSA) to the list of known hosts.
bind [::1]:8080: Cannot assign requested address
WARNING:

To increase the performance of the tunnel, consider installing NumPy. For instructions,
please see https://cloud.google.com/iap/docs/using-tcp-forwarding#increasing_the_tcp_upload_bandwidth

bind [::1]:8080: Cannot assign requested address
Linux w261-m 5.10.0-0.bpo.15-amd64 #1 SMP Debian 5.10.120-1~bpo10+1 (2022-06-13) x86_64
```

CLOUD SHELL

Terminal (w261-student-360302) ×

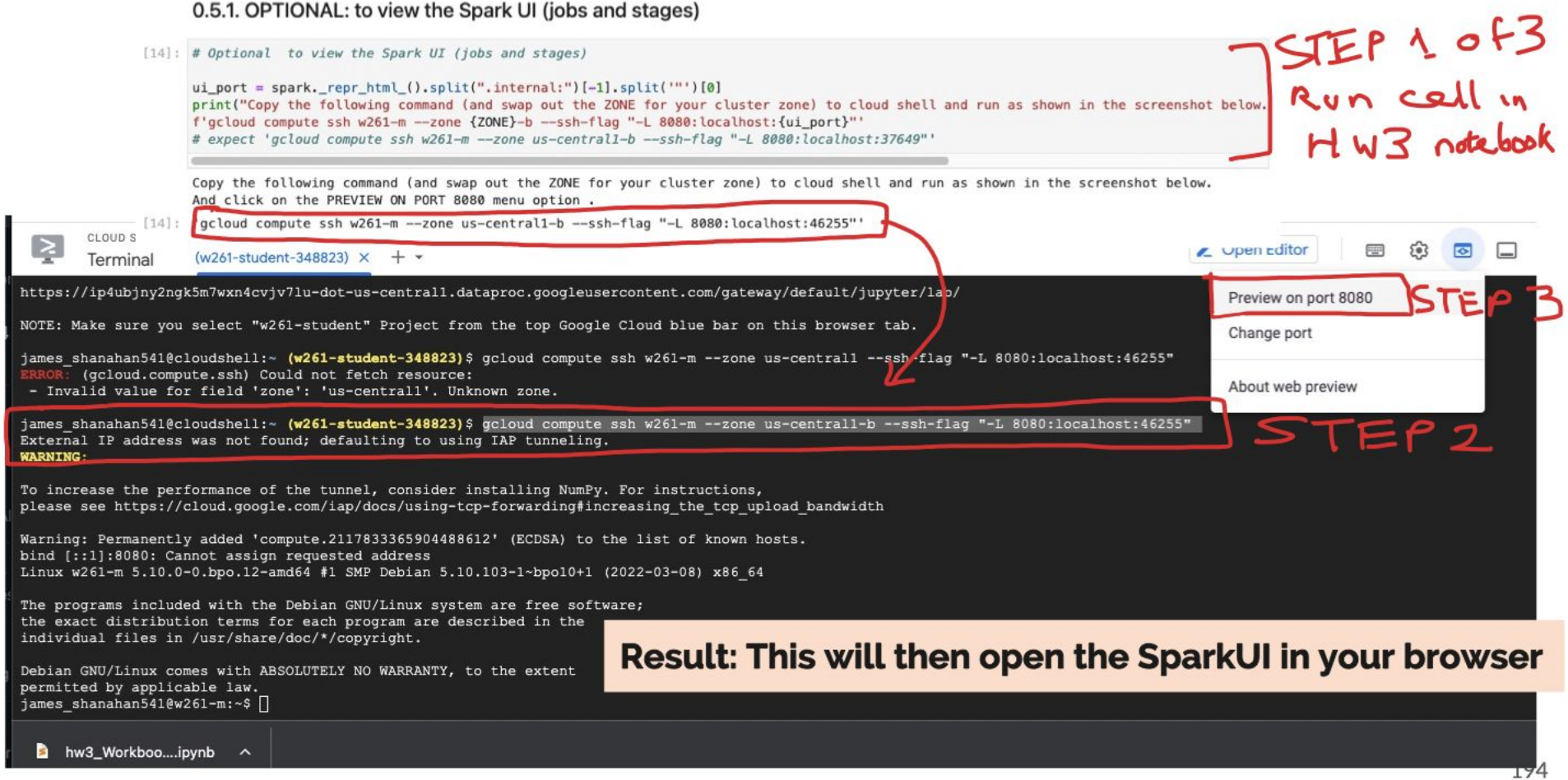
Privacy Policy · Terms of Service

CLOUD SHELL

Terminal (w261-student-360302) ×

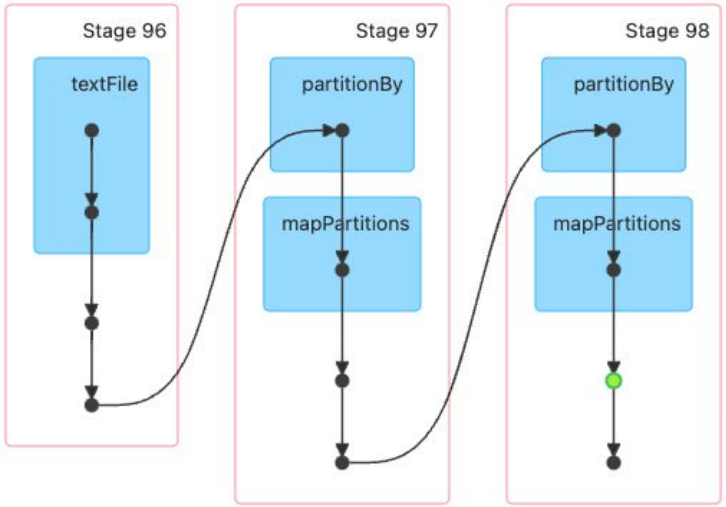
```
NOTE: In order to be able to see the SPARK UI, you need to have a running Notebook with an active Spark Context.
james_shanahan@cloudshell:~ (w261-student-360302)$ gcloud compute ssh w261-m --zone us-central1-a --ssh-flag "-L 8080:localhost:404
WARNING: The private SSH key file for gcloud does not exist.
WARNING: The public SSH key file for gcloud does not exist.
WARNING: You do not have an SSH key for gcloud.
WARNING: SSH keygen will be executed to generate a key.
This tool needs to create the directory [/home/james_shanahan/.ssh] before being able to generate SSH keys.
|.+ o *          |
|+o* O .         |
|*+.B . .        |
|++* B . S       |
|.+oB *          |
|.o+oE           |
|oo+=.o          |
|.o=+=oo         |
+----[SHA256]-----+
External IP address was not found; defaulting to using IAP tunneling.
Updating project ssh metadata...working..Updated [https://www.googleapis.com/compute/v1/projects/w261-student-360302].
Updating project ssh metadata...done.
Waiting for SSH key to propagate.
WARNING:

To increase the performance of the tunnel, consider installing NumPy. For instructions,
please see https://cloud.google.com/iap/docs/using-tcp-forwarding#increasing_the_tcp_upload_bandwidth

Warning: Permanently added 'compute.7499392942176644213' (ECDSA) to the list of known hosts.
bind [::1]:8080: Cannot assign requested address
WARNING:

To increase the performance of the tunnel, consider installing NumPy. For instructions,
please see https://cloud.google.com/iap/docs/using-tcp-forwarding#increasing_the_tcp_upload_bandwidth

bind [::1]:8080: Cannot assign requested address
Linux w261-m 5.10.0-0.bpo.15-amd64 #1 SMP Debian 5.10.120-1~bpo10+1 (2022-06-13) x86_64
```

To access Spark UI: after your launch Spark on DataProc do:



Result: This will then open the SparkUI in your browser

▾ DAG Visualization



Spark 3.1.2 | Jobs | Stages | Storage | Environment

hw3_notebook application UI

Details for Job 41

Status: SUCCEEDED
Submitted: 2022/05/31 04:44:26
Duration: 13 min
Completed Stages: 3

▸ Event Timeline
▸ DAG Visualization

▾ Completed Stages (3)

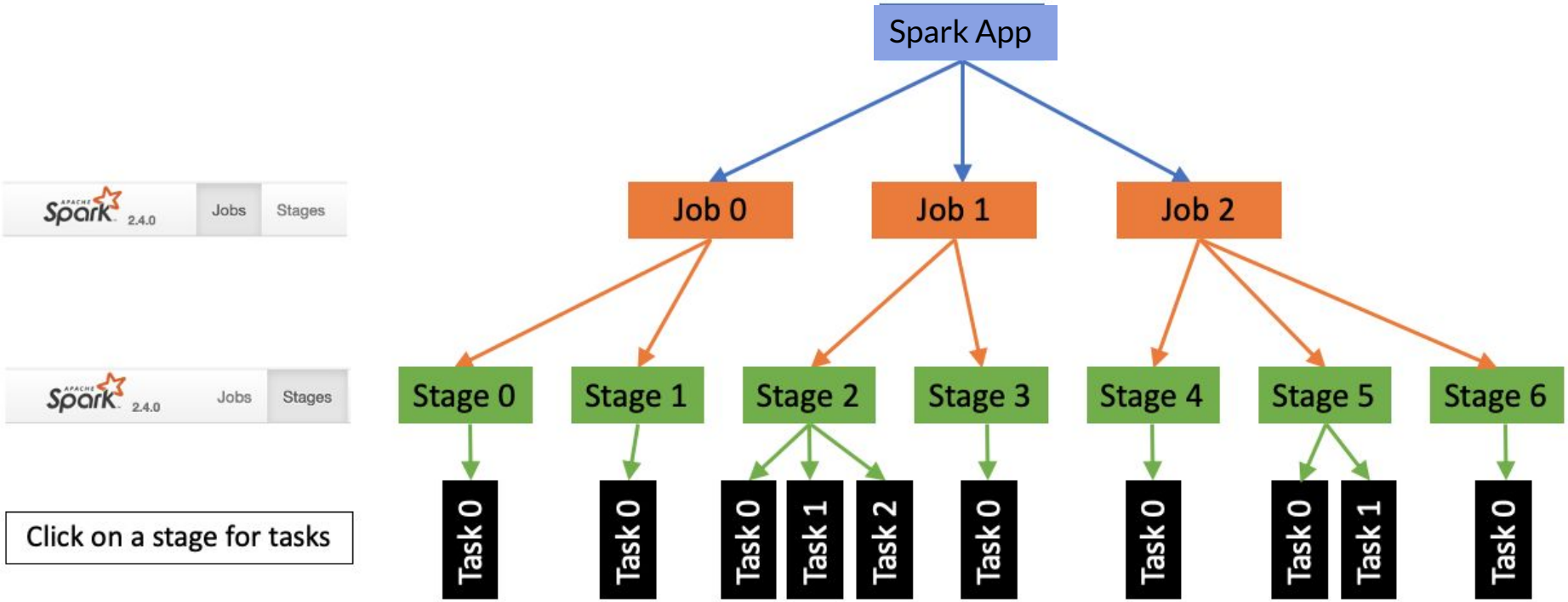
Page: 1

1 Pages. Jump to 1 . Show 100 items in a page. Go

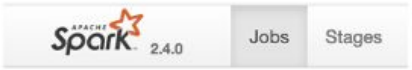
Stage Id ▾	Pool Name	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
98	default	takeOrdered at /tmp/ipykernel_10558/1055424797.py:26 +details	2022/05/31 04:57:34	6 s	190/190			96.0 MiB	
97	default	reduceByKey at /tmp/ipykernel_10558/1055424797.py:20 +details	2022/05/31 04:55:49	1.7 min	190/190			1642.5 MiB	96.0 MiB
96	default	distinct at /tmp/ipykernel_10558/1055424797.py:20 +details	2022/05/31 04:44:26	11 min	190/190	689.8 MiB			1642.5 MiB

Page: 1

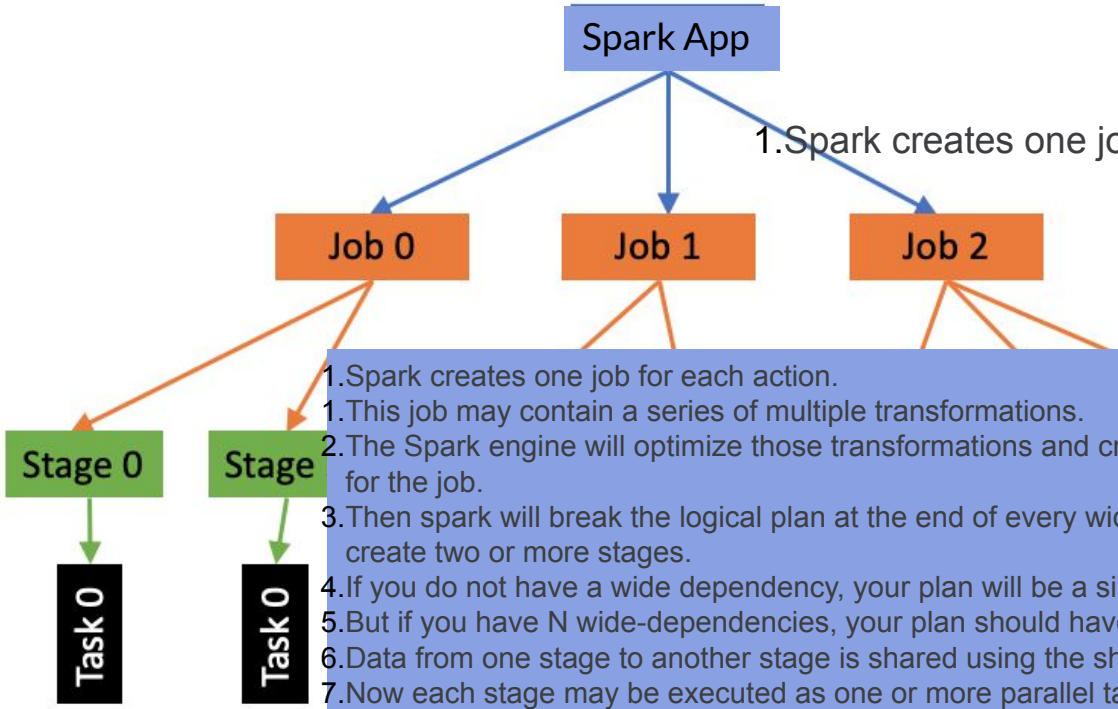
1 Pages. Jump to 1 . Show 100 items in a page. Go



Source: https://medium.com/data-engineering-on-cloud/advance-spark-concepts-for-job-interview-part-1-b7c2cadffc42



Click on a stage for tasks



1. Spark creates one job for each action.

1. Spark creates one job for each action.
1. This job may contain a series of multiple transformations.
2. The Spark engine will optimize those transformations and creates a logical plan for the job.
3. Then spark will break the logical plan at the end of every wide dependency and create two or more stages.
4. If you do not have a wide dependency, your plan will be a single-stage plan.
5. But if you have N wide-dependencies, your plan should have N+1 stages.
6. Data from one stage to another stage is shared using the shuffle/sort operation.
7. Now each stage may be executed as one or more parallel tasks.
8. The number of tasks in the stage is equal to the number of input partitions.

Source: https://medium.com/data-engineering-on-cloud/advance-spark-concepts-for-job-interview-part-1-b7c2cadffc42

An Apache Spark app can have many jobs (actions)

An Apache Spark App

- An app may consist of a pipeline (or multiple pipelines) of operations (transformations and actions)

Job

- Spark creates one job for each action.
- This job may contain a series of multiple transformations.
- The Spark engine will optimize those transformations and creates a logical plan for the job.

Stages

- Then spark will break the logical plan at the end of every wide dependency and create two or more stages.
- If you do not have a wide dependency, your plan will be a single-stage plan.
- But if you have N wide-dependencies, your plan should have N+1 stages.
- Data from one stage to another stage is shared using the shuffle/sort operation.
  - Now each stage may be executed as one or more parallel tasks.
  - The number of tasks in the stage is equal to the number of input partitions.

Adopted from : https://medium.com/data-engineering-on-cloud/advance-spark-concepts-for-job-interview-part-1-b7c2cadffc42

Save Spark RDD to disk (notice PART-00000 - PART-00001 etc..!)

8.0.3.1. Checkpoint the stripes (Just RUN all cells in this section AS IS and review outputs)

Let's save your full stripes to disk. Then we can reload later if needed. We repartition our data first and then save (as otherwise, we will end up with 190 partitions; I wonder why!).

```
!gsutil -m rm -r {HW3_FOLDER}/stripes 2> /dev/null   ##remove old results
stripesRDD.repartition(4).saveAsTextFile(f'{HW3_FOLDER}/stripes')  #repartition and write partitions to Google Cloud Bucket
!gsutil ls -lh {HW3_FOLDER}/stripes
```

The above produces the following output directory:

```
       0 B  2022-05-31T05:35:02Z  gs://w261-jgs/main/Assignments/HW3/docker/student/stripes/
       0 B  2022-05-31T05:35:03Z  gs://w261-jgs/main/Assignments/HW3/docker/student/stripes/_SUCCESS
  1.74 MiB  2022-05-31T05:35:02Z  gs://w261-jgs/main/Assignments/HW3/docker/student/stripes/part-00000
   1.6 MiB  2022-05-31T05:35:01Z  gs://w261-jgs/main/Assignments/HW3/docker/student/stripes/part-00001
  1.58 MiB  2022-05-31T05:35:02Z  gs://w261-jgs/main/Assignments/HW3/docker/student/stripes/part-00002
  1.52 MiB  2022-05-31T05:35:01Z  gs://w261-jgs/main/Assignments/HW3/docker/student/stripes/part-00003
TOTAL: 6 objects, 6745577 bytes (6.43 MiB)
```

The following code displays the stripe of cooccurence words for the term `sea`:

```
!gsutil cat {HW3_FOLDER}/stripes/part-00000|head -n 1
```

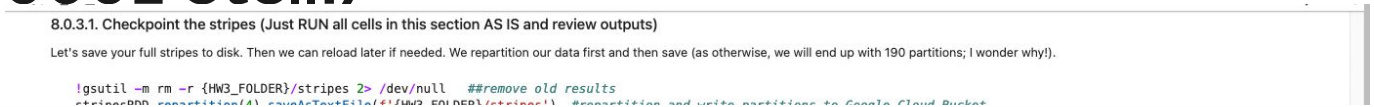
```
('sea', {'sweeping', 'twisted', 'athenians', 'fog', 'tumult', 'repression', 'morphology', 'jane', 'secreted', 'tents', 'barred', 'sadness', 'hamlet', 'turbulent',
'rains', 'robe', 'imagery', 'myths', 'orient', 'intervening', 'victories', 'accumulate', 'sinners', 'constancy', 'strained', 'sermons', 'shoe', 'trembled',
'merged', 'eastward', 'avoidance', 'sensibility', 'informing', 'silently', 'dip', 'surround', 'blocked', 'voyages', 'bursting', 'vastly', 'southeastern', 'cracks',
'tore', 'temperament', "ship's", 'odor', 'atlas', 'matthew', 'ether', 'colonization', 'irresistible', 'shells', 'alaska', 'gaza', 'distributions', 'farthest',
'silly', 'flush', 'ugly', 'transparent', 'arabian', 'sandy', 'steering', 'penetrating', 'burns', 'norway', 'thames', 'moonlight', 'plunge', 'beset', 'yielding',
'tuesday', 'impacts', 'cheese', 'convex', 'armistice', 'polished', 'freshness', 'belgium', 'saturation', 'dumb', 'spoil', 'shines', 'sunset', 'softly', 'laden',
'realms', 'alexandria', 'parallels', 'weep', 'ushered', 'violently', 'expanse', 'travellers', 'insoluble', 'downs', 'roofs', 'filtered', 'ashore', 'graces',
'obscured', 'establishments', 'traversed', 'crystalline', 'warmer', 'skins', 'viewing', 'fascination', 'liverpool', 'contamination', 'sails', 'masculine', 'usages',
'bucket', 'dipped', 'dew', 'fare', 'overlooking', 'necks', 'sticks', 'weighing', 'danube', 'mast', 'phosphorus', 'mate', 'attested', 'anonymous', 'wax',
'finishing', 'parked', 'flocks', 'humidity', 'endurance', 'terrors', 'carpet', 'misfortunes', 'hydroxide', 'crazy', 'priesthood', 'hungary', 'nova', 'believeth',
'remotest', 'occupants', 'complexion', 'floors', 'stationary', 'provoked', 'osmotic', 'spoils', 'clearance', 'hangs', 'openings', 'halfway', 'inorganic', 'nursery',
'vigilance', 'conqueror', 'ft', 'feathers', 'roses', 'emblem', 'lawn', 'damp', 'switzerland', 'drinks', 'contradictory', 'drained', 'ordinances', 'captains',
'barren', 'steamer', 'pursuits', 'storms', 'wasting', 'frankly', 'sequences', 'pitched', 'aggravated', 'viceroy', 'leaped', 'cunning', 'simon', 'marching', 'lends',
'sherman', 'centered', 'genome', 'iran', 'sued', 'imputed', 'perilous', 'desperately', 'southward', 'maiden', 'unusually', 'crosses', 'revealing', 'uppermost',
'remission', 'inherit', 'sunny', 'ink', 'restless', 'lighting', 'serpent', 'scarlet', 'hebrews', 'flourish', 'terminology', 'bidding', 'autobiography', 'despise',
'signification', 'preparatory', 'radioactive', 'drying', 'persia', 'unfamiliar', 'twist', 'fiery', 'boon', 'delights', 'commonest', 'bounty', 'traders', 'whoever'})
```

[85]:
```
# part d - save your full stripes to file for ease of retrival later... Please run code as is
!gsutil -m rm -r {HW3_FOLDER}/stripes 2> /dev/null   ##remove old results
stripesRDD.repartition(4).saveAsTextFile(f'{HW3_FOLDER}/stripes')  #repartition and write partitions to Google Cloud Bucket
```

[87]:
```
# part d - list all partitions in the saved output folder... (RUN THIS CELL AS IS)
!gsutil ls -lh {HW3_FOLDER}/stripes
```

```
       0 B  2022-05-31T05:35:02Z  gs://w261-jgs/main/Assignments/HW3/docker/student/stripes/
       0 B  2022-05-31T05:35:03Z  gs://w261-jgs/main/Assignments/HW3/docker/student/stripes/_SUCCESS
  1.74 MiB  2022-05-31T05:35:02Z  gs://w261-jgs/main/Assignments/HW3/docker/student/stripes/part-00000
   1.6 MiB  2022-05-31T05:35:01Z  gs://w261-jgs/main/Assignments/HW3/docker/student/stripes/part-00001
  1.58 MiB  2022-05-31T05:35:02Z  gs://w261-jgs/main/Assignments/HW3/docker/student/stripes/part-00002
  1.52 MiB  2022-05-31T05:35:01Z  gs://w261-jgs/main/Assignments/HW3/docker/student/stripes/part-00003
TOTAL: 6 objects, 6745577 bytes (6.43 MiB)
```

Save Spark RDD to disk (notice PART-00000 - PART-00001 etc..!)



8.0.3.1. Checkpoint the stripes (Just RUN all cells in this section AS IS and review outputs)

Let's save your full stripes to disk. Then we can reload later if needed. We repartition our data first and then save (as otherwise, we will end up with 190 partitions; I wonder why!).

```
!gsutil -m rm -r {HW3_FOLDER}/stripes 2> /dev/null   ##remove old results
stripesRDD.repartition(4).saveAsTextFile(f'{HW3_FOLDER}/stripes')  #repartition and write partitions to Google Cloud Bucket
!gsutil ls -lh {HW3_FOLDER}/stripes
```

The above produces the following output directory:

```
       0 B  2022-05-31T05:35:02Z  gs://w261-jgs/main/Assignments/HW3/docker/student/stripes/
       0 B  2022-05-31T05:35:03Z  gs://w261-jgs/main/Assignments/HW3/docker/student/stripes/_SUCCESS
  1.74 MiB  2022-05-31T05:35:02Z  gs://w261-jgs/main/Assignments/HW3/docker/student/stripes/part-00000
   1.6 MiB  2022-05-31T05:35:01Z  gs://w261-jgs/main/Assignments/HW3/docker/student/stripes/part-00001
  1.58 MiB  2022-05-31T05:35:02Z  gs://w261-jgs/main/Assignments/HW3/docker/student/stripes/part-00002
  1.52 MiB  2022-05-31T05:35:01Z  gs://w261-jgs/main/Assignments/HW3/docker/student/stripes/part-00003
TOTAL: 6 objects, 6745577 bytes (6.43 MiB)
```

The following code displays the stripe of cooccurence words for the term `sea` :

```
!gsutil cat {HW3_FOLDER}/stripes/part-00000|head -n 1
```

```
('sea', {'sweeping', 'twisted', 'athenians', 'fog', 'tumult', 'repression', 'morphology', 'jane', 'secreted', 'tents', 'barred', 'sadne
```


       0 B 2022-05-31T05:35:02Z gs://w261-jgs/main/Assignments/HW3/docker/student/stripes/
       0 B 2022-05-31T05:35:03Z gs://w261-jgs/main/Assignments/HW3/docker/student/stripes/_SUCCESS
  1.74 MiB 2022-05-31T05:35:02Z gs://w261-jgs/main/Assignments/HW3/docker/student/stripes/part-00000
   1.6 MiB 2022-05-31T05:35:01Z gs://w261-jgs/main/Assignments/HW3/docker/student/stripes/part-00001
  1.58 MiB 2022-05-31T05:35:02Z gs://w261-jgs/main/Assignments/HW3/docker/student/stripes/part-00002
  1.52 MiB 2022-05-31T05:35:01Z gs://w261-jgs/main/Assignments/HW3/docker/student/stripes/part-00003
TOTAL: 6 objects, 6745577 bytes (6.43 MiB)

Housekeeping

MidTerm Course Evaluation (week)

Please fill out the course evals over the next 24 possible) here. These are important for the course and for the program. Thank you!!

https://course- ions.berkeley.edu/berkeley/

Please complete asap! PLEASE

Take 10 minutes to complete now (due this week)

Quick review

Introduction to Spark With RDDs: Part II

5: Introduction to Spark With RDDs: Part II

1h 14m Total Video Time

Course Content	Description
5.1 Weekly Introduction 5	Sequence | 1m | View Sequence Outline
5.2 Aggregations	Sequence | 7m 23s | View Sequence Outline
5.3 Controlling Partitions	Sequence | 6m 36s | View Sequence Outline
5.4 Monitoring and Debugging	Sequence | 23m | View Sequence Outline
5.5 Performance Tuning	Sequence | 14m 21s | View Sequence Outline
5.6 Clustering Overview	Interactive Video | 7m 4s
5.7 K-Means Algorithm	Interactive Video | 7m 25s
5.8 K-Means at Scale	Sequence | 6m 6s | View Sequence Outline
5.9 Summary	Sequence | 47s | View Sequence Outline

Week 4

Advanced

4.11-4.13

4: Introduction to Spark With RDDs: Part I

2h 20m Total Video Time

Course Content	Description
4.1 Weekly Introduction 4	Sequence | 1m 33s | View Sequence Outline
4.2 Background on Spark	Interactive Video | 14m 30s
4.3 Functional Programming Review	Interactive Video | 7m 46s
4.4 Spark Basics	Interactive Video | 20m 24s
4.5 Programming With Base RDDs	Interactive Video | 21m 42s
4.6 Quiz 4-1	Sequence | View Sequence Outline
4.7 Animated Example: Data Flow	Interactive Video | 7m 58s
4.8 Pair RDDs	Interactive Video | 13m 13s
4.9 Quiz 4-2	Sequence | View Sequence Outline
4.10 Basic Word Count in Spark	Interactive Video | 11m 34s
4.11 Pairs and Stripes	Sequence | 13m 7s | View Sequence Outline
4.12 Relative Frequencies Revisited	Sequence | 11m 7s | View Sequence Outline
4.13 Inverted Index	Sequence | 12m 4s | View Sequence Outline
4.14 Spark Summary	Interactive Video | 5m 31s

Week 5

5: Introduction to Spark With RDDs: Part II

1h 14m Total Video Time

Course Content	Description
5.1 Weekly Introduction 5	Sequence | 1m | View Sequence Outline
5.2 Aggregations	Sequence | 7m 23s | View Sequence Outline
5.3 Controlling Partitions	Sequence | 6m 36s | View Sequence Outline
5.4 Monitoring and Debugging	Sequence | 23m | View Sequence Outline
5.5 Performance Tuning	Sequence | 14m 21s | View Sequence Outline
5.6 Clustering Overview	Interactive Video | 7m 4s
5.7 K-Means Algorithm	Interactive Video | 7m 25s
5.8 K-Means at Scale	Sequence | 6m 6s | View Sequence Outline
5.9 Summary	Sequence | 47s | View Sequence Outline

HW3 Due

Week	Topic	Deadlines
1	Intro to Machine Learning at Scale	
2	Parallel Computation Frameworks	HW 1 due Sunday midnight at the end of week 2
3	Map-Reduce Algorithm Design	
4	Intro to Spark/Map-Reduce with RDDs (part 1)	HW 2 due Sunday midnight at the end of this week
5	Intro to Spark/Map-Reduce with RDDs (part 2)	
6	Distributed Supervised ML (part 1)	HW 3 due Sunday midnight at the end of this week
7	Distributed Supervised ML (part 2)	
8	Big Data Systems and Pipelines	HW 4 due Sunday midnight at the end of this week
9	Graph Algorithms at Scale (part 1)	
10	Graph Algorithms at Scale (part 2)	

HW 3 - Synonym Detection In Spark

`MIDS w261: Machine Learning at Scale | UC Berkeley School of Information | Fall 2018`

In the last homework assignment you performed Naive Bayes to classify documents as 'ham' or 'spam.' In doing so, we relied on the implicit assumption that the list of words in a document can tell us something about the nature of that document's content. We'll rely on a similar intuition this week: the idea that, if we analyze a large enough corpus of text, the list of words that appear in small window before or after a vocabulary term can tell us something about that term's meaning. This is similar to the intuition behind the word2vec algorithm.

This will be your first assignment working in Spark. You'll perform Synonym Detection by repurposing an algorithm commonly used in Natural Language Processing to perform document similarity analysis. In doing so you'll also become familiar with important datatypes for efficiently processing sparse vectors and a number of set similarity metrics (e.g. Cosine, Jaccard, Dice). By the end of this homework you should be able to:

- ... **define** the terms `one-hot encoding`, `co-occurrance matrix`, `stripe`, `inverted index`, `postings`, and `basis vocabulary` in the context of both synonym detection and document similarity analysis.
- ... **explain** the reasoning behind using a word stripe to compare word meanings.
- ... **identify** what makes set-similarity calculations computationally challenging.
- ... **implement** stateless algorithms in Spark to build stripes, inverted index and compute similarity metrics.
- ... **identify** when it makes sense to take a stripe approach and when to use pairs
- ... **apply** appropriate metrics to assess the performance of your synonym detection algorithm.

RECOMMENDED READING FOR HW3:
Your reading assignment for weeks 4 and 5 were fairly heavy and you may have glossed over the papers on dimension independent similarity metrics by Zadeh et al and pairwise document similarity by Elsayed et al. If you haven't already, this would be a good time to review those readings, especially when it comes to the similarity formulas -- they are directly relevant to this assignment.

DITP Chapter 4 - Inverted Indexing for Text Retrieval. While this text is specific to Hadoop, the Map/Reduce concepts still apply.

Please refer to the `README` for homework submission instructions and additional resources.

Notebook Set Up

Grab the HW3 data

```
# HW3 data
# 3-4 meg compressed
!gsutil ls -l gs://w261-hw-data/notebooks/jupyter/main/Assignments/HW3/data/*
```

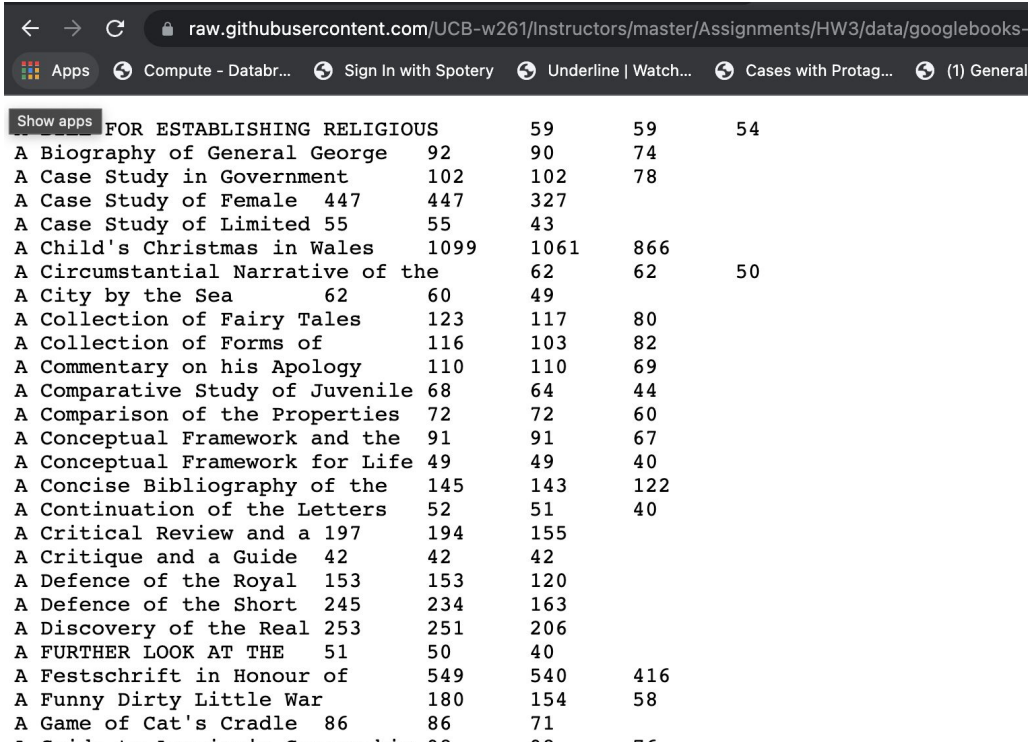
1.2. HW 3 data ¶

13]:

```
# HW
# 3-4 meg compressed
!gsutil ls -l gs://w261-hw-data/notebooks/jupyter/main/Assignments/HW3/data/*
```

```
3838591  2022-08-22T14:12:35Z  gs://w261-hw-data/notebooks/jupyter/main/Assignments/HW3/data/googlebooks-eng-all-5gram-20090715-0-filtered.txt.gz
     70  2022-08-22T14:12:35Z  gs://w261-hw-data/notebooks/jupyter/main/Assignments/HW3/data/googlebooks-eng-all-5gram-20090715-1-filtered.txt.gz
3841798  2022-08-22T14:12:35Z  gs://w261-hw-data/notebooks/jupyter/main/Assignments/HW3/data/googlebooks-eng-all-5gram-20090715-10-filtered.txt.gz
3850883  2022-08-22T14:12:35Z  gs://w261-hw-data/notebooks/jupyter/main/Assignments/HW3/data/googlebooks-eng-all-5gram-20090715-100-filtered.txt.gz
3849115  2022-08-22T14:12:35Z  gs://w261-hw-data/notebooks/jupyter/main/Assignments/HW3/data/googlebooks-eng-all-5gram-20090715-101-filtered.txt.gz
3826854  2022-08-22T14:12:35Z  gs://w261-hw-data/notebooks/jupyter/main/Assignments/HW3/data/googlebooks-eng-all-5gram-20090715-102-filtered.txt.gz
3850549  2022-08-22T14:12:35Z  gs://w261-hw-data/notebooks/jupyter/main/Assignments/HW3/data/googlebooks-eng-all-5gram-20090715-103-filtered.txt.gz
3832928  2022-08-22T14:12:35Z  gs://w261-hw-data/notebooks/jupyter/main/Assignments/HW3/data/googlebooks-eng-all-5gram-20090715-104-filtered.txt.gz
3851276  2022-08-22T14:12:35Z  gs://w261-hw-data/notebooks/jupyter/main/Assignments/HW3/data/googlebooks-eng-all-5gram-20090715-105-filtered.txt.gz
3847239  2022-08-22T14:12:35Z  gs://w261-hw-data/notebooks/jupyter/main/Assignments/HW3/data/googlebooks-eng-all-5gram-20090715-106-filtered.txt.gz
3852765  2022-08-22T14:12:35Z  gs://w261-hw-data/notebooks/jupyter/main/Assignments/HW3/data/googlebooks-eng-all-5gram-20090715-107-filtered.txt.gz
3846105  2022-08-22T14:12:35Z  gs://w261-hw-data/notebooks/jupyter/main/Assignments/HW3/data/googlebooks-eng-all-5gram-20090715-108-filtered.txt.gz
3838061  2022-08-22T14:12:35Z  gs://w261-hw-data/notebooks/jupyter/main/Assignments/HW3/data/googlebooks-eng-all-5gram-20090715-109-filtered.txt.gz
3855597  2022-08-22T14:12:35Z  gs://w261-hw-data/notebooks/jupyter/main/Assignments/HW3/data/googlebooks-eng-all-5gram-20090715-11-filtered.txt.gz
3841600  2022-08-22T14:12:35Z  gs://w261-hw-data/notebooks/jupyter/main/Assignments/HW3/data/googlebooks-eng-all-5gram-20090715-110-filtered.txt.gz
3842125  2022-08-22T14:12:35Z  gs://w261-hw-data/notebooks/jupyter/main/Assignments/HW3/data/googlebooks-eng-all-5gram-20090715-111-filtered.txt.gz
3847993  2022-08-22T14:12:35Z  gs://w261-hw-data/notebooks/jupyter/main/Assignments/HW3/data/googlebooks-eng-all-5gram-20090715-112-filtered.txt.gz
3843500  2022-08-22T14:12:35Z  gs://w261-hw-data/notebooks/jupyter/main/Assignments/HW3/data/googlebooks-eng-all-5gram-20090715-113-filtered.txt.gz
3852260  2022-08-22T14:12:35Z  gs://w261-hw-data/notebooks/jupyter/main/Assignments/HW3/data/googlebooks-eng-all-5gram-20090715-114-filtered.txt.gz
3861740  2022-08-22T14:12:35Z  gs://w261-hw-data/notebooks/jupyter/main/Assignments/HW3/data/googlebooks-eng-all-5gram-20090715-115-filtered.txt.gz
```

A closer at one of the data files



Pairs and Stripes in Spark [advanced]

Document similarity vs synonym detection
“pattern” vs “representation”
Dense vs sparse representation
Inverted Index & Postings
Basis Vocabulary

Week 4

4: Introduction to Spark With RDDs: Part I

2h 20m Total Video Time

Course Content	Description
4.1 Weekly Introduction 4	Sequence | 1m 33s | View Sequence Outline
4.2 Background on Spark	Interactive Video | 14m 30s
4.3 Functional Programming Review	Interactive Video | 7m 46s
4.4 Spark Basics	Interactive Video | 20m 24s
4.5 Programming With Base RDDs	Interactive Video | 21m 42s
4.6 Quiz 4-1	Sequence | View Sequence Outline
4.7 Animated Example: Data Flow	Interactive Video | 7m 58s
4.8 Pair RDDs	Interactive Video | 13m 13s
4.9 Quiz 4-2	Sequence | View Sequence Outline
4.10 Basic Word Count in Spark	Interactive Video | 11m 34s
4.11 Pairs and Stripes	Sequence | 13m 7s | View Sequence Outline
4.12 Relative Frequencies Revisited	Sequence | 11m 7s | View Sequence Outline
4.13 Inverted Index	Sequence | 12m 4s | View Sequence Outline
4.14 Spark Summary	Interactive Video | 5m 31s

4:12

Week 5

5: Introduction to Spark With RDDs: Part II

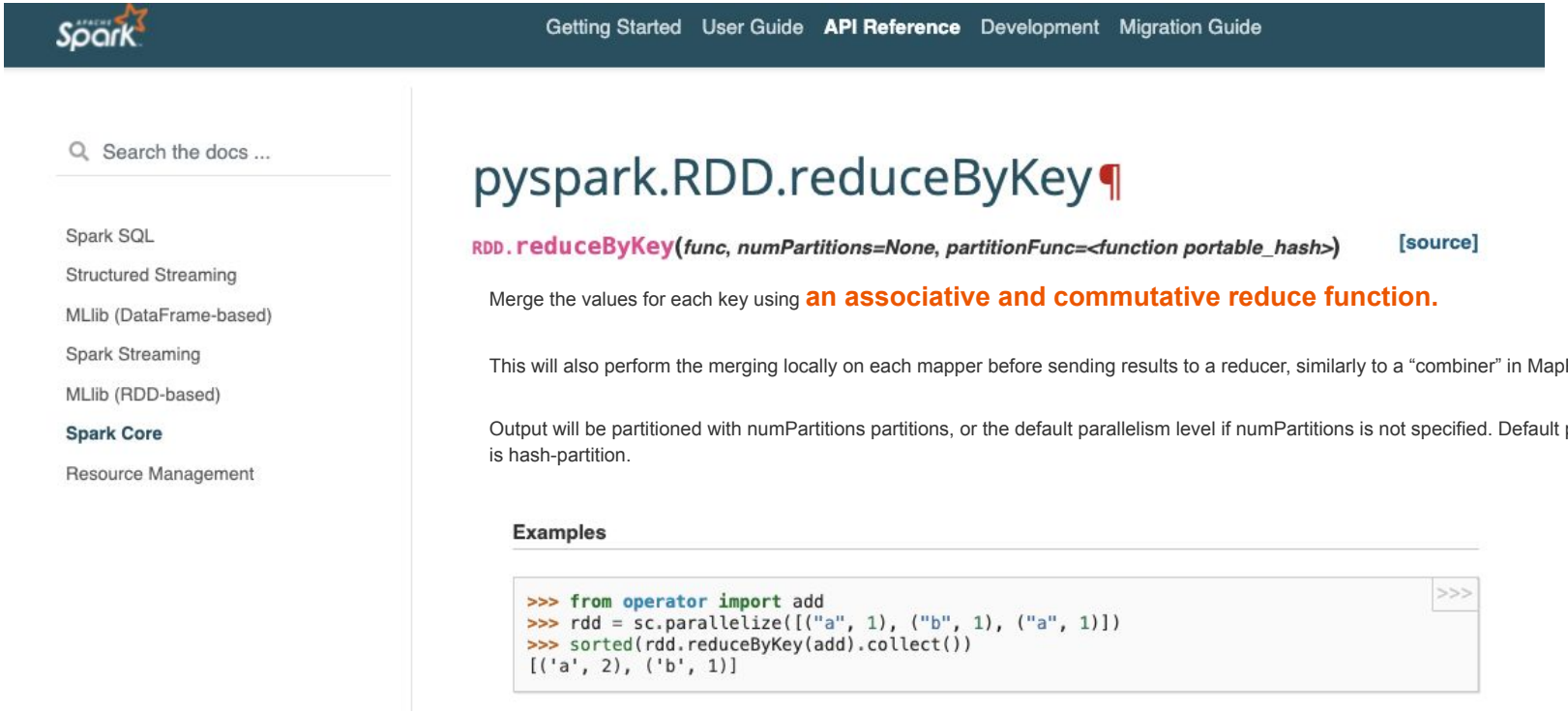
1h 14m Total Video Time

Course Content	Description
5.1 Weekly Introduction 5	Sequence | 1m | View Sequence Outline
5.2 Aggregations	Sequence | 7m 23s | View Sequence Outline
5.3 Controlling Partitions	Sequence | 6m 36s | View Sequence Outline
5.4 Monitoring and Debugging	Sequence | 23m | View Sequence Outline
5.5 Performance Tuning	Sequence | 14m 21s | View Sequence Outline
5.6 Clustering Overview	Interactive Video | 7m 4s
5.7 K-Means Algorithm	Interactive Video | 7m 25s
5.8 K-Means at Scale	Sequence | 6m 6s | View Sequence Outline
5.9 Summary	Sequence | 47s | View Sequence Outline

Calculate the relative frequency of bi-grams (pairs)

```
DATA = sc.parallelize(['dog aardvark pig banana',
                       'bear zebra pig'])
```

```
[
    [
    ('bear - pig', 0.5),
    ('bear - zebra', 0.5),
    ('dog - aardvark', 0.33),
    ('dog - banana', 0.33),
    ('dog - pig', 0.33),
    ('pig - banana', 1.0)
    ], [
    ('aardvark - banana', 0.5),
    ('aardvark - pig', 0.5),
    ('zebra - pig', 1.0)
```



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

Which API Call Causes Most Tickets?

map
filter
groupBy
sort
union
join
leftOuterJoin
rightOuterJoin

reduce
count
fold
reduceByKey
groupByKey
cogroup
cross
zip

sample
take
first
partitionBy
mapWith
pipe
save
...

Example Problem

```
pairs = data.map(word => (word, 1))

groups = pairs.groupByKey()

groups.map((k, vs) => (k, vs.sum))
```

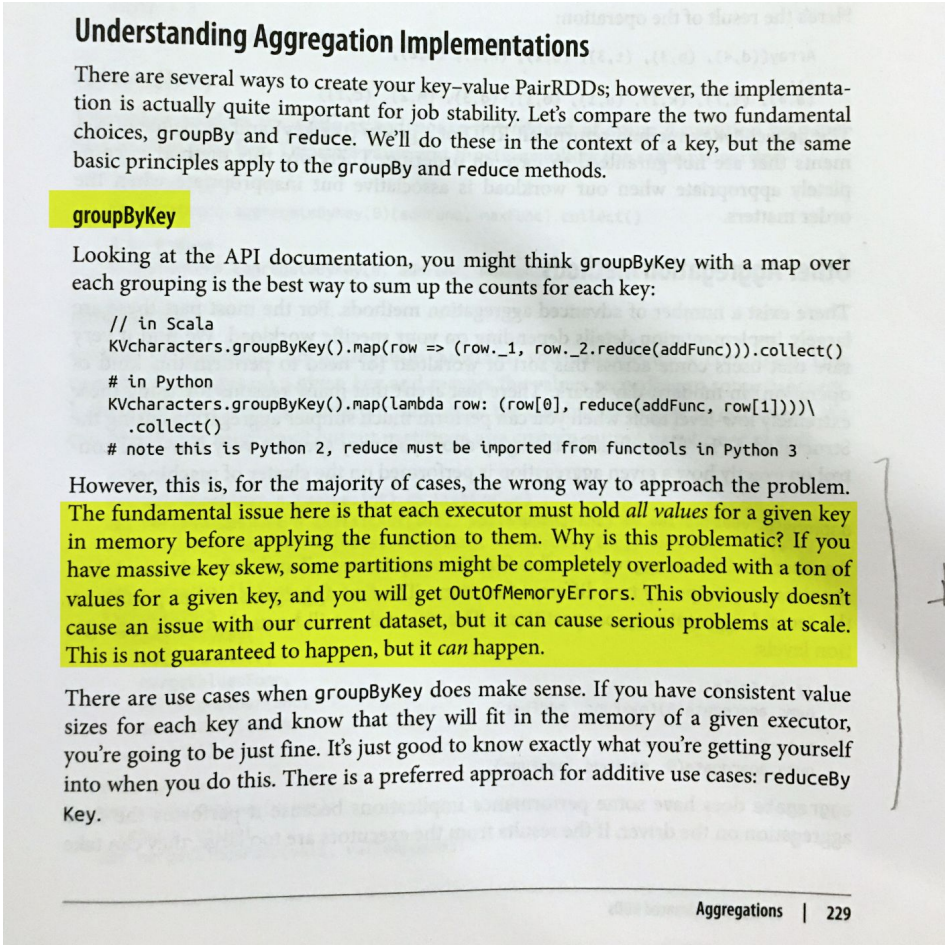
Materializes all groups as Seq[Int] objects

Then promptly aggregates them

Solution

How would you rewrite the above word-count problem?

When would you want to use groupByKey?



Understanding Aggregation Implementations

There are several ways to create your key–value PairRDDs; however, the implementation is actually quite important for job stability. Let's compare the two fundamental choices, `groupBy` and `reduce`. We'll do these in the context of a key, but the same basic principles apply to the `groupBy` and `reduce` methods.

groupByKey

Looking at the API documentation, you might think `groupByKey` with a map over each grouping is the best way to sum up the counts for each key:

```
// in Scala
KVcharacters.groupByKey().map(row => (row._1, row._2.reduce(addFunc))).collect()

# in Python
KVcharacters.groupByKey().map(lambda row: (row[0], reduce(addFunc, row[1])))\
  .collect()
# note this is Python 2, reduce must be imported from functools in Python 3
```

However, this is, for the majority of cases, the wrong way to approach the problem. The fundamental issue here is that each executor must hold *all values* for a given key in memory before applying the function to them. Why is this problematic? If you have massive key skew, some partitions might be completely overloaded with a ton of values for a given key, and you will get `OutOfMemoryErrors`. This obviously doesn't cause an issue with our current dataset, but it can cause serious problems at scale. This is not guaranteed to happen, but it *can* happen.

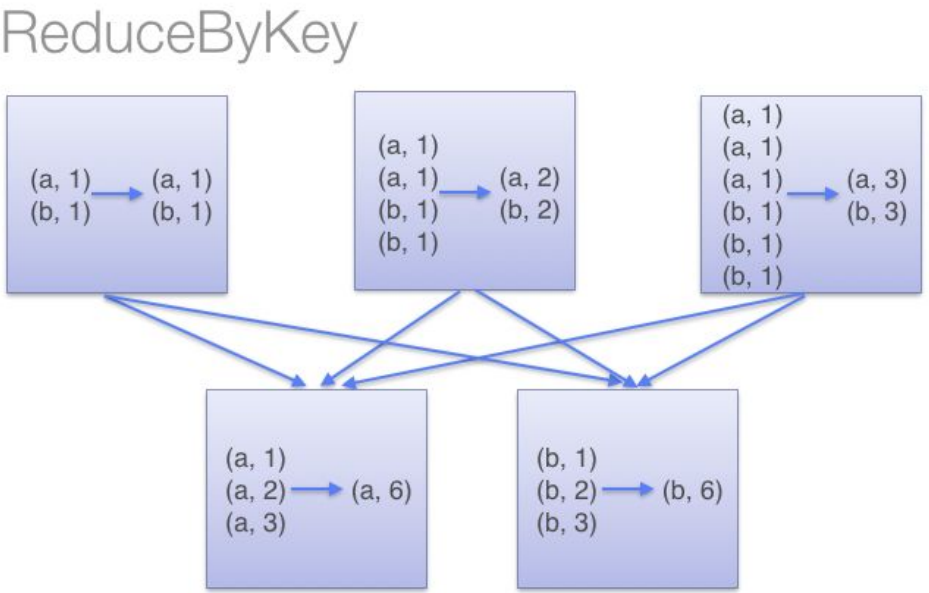
There are use cases when `groupByKey` does make sense. If you have consistent value sizes for each key and know that they will fit in the memory of a given executor, you're going to be just fine. It's just good to know exactly what you're getting yourself into when you do this. There is a preferred approach for additive use cases: `reduceBy Key`.

ReduceByKey

While both the `groupByKey()` and `reduceByKey()` transformations can often be used to solve the same problem and will produce the same answer, the `reduceByKey()` transformation works much better for large distributed datasets. This is because Spark knows it can combine output with a common key on each partition *before* shuffling (redistributing) the data across nodes. Only use `groupByKey()` if the operation would not benefit from reducing the data before the shuffle occurs.

Look at the diagram below to understand how `reduceByKey` works. Notice how pairs on the same machine with the same key are combined (by using the lamdba function passed into reduceByKey) before the data is shuffled. Then the lamdba function is called again to reduce all the values from each partition to produce one final result.

https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/4538883350836412/3508262507098993/7991902709591605/latest.html

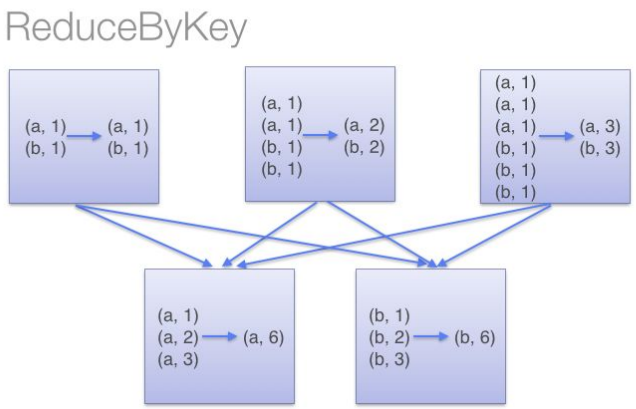


(6b) `groupByKey` and `reduceByKey`

Let's investigate the additional transformations: groupByKey() and reduceByKey().

Both of these transformations operate on pair RDDs. A pair RDD is an RDD where each element is a pair tuple (key, value). For example, `sc.parallelize([('a', 1), ('a', 2), ('b', 1)])` would create a pair RDD where the keys are 'a', 'a', 'b' and the values are 1, 2, 1. The `reduceByKey()` transformation gathers together pairs that have the same key and applies a function to two associated values at a time. `reduceByKey()` operates by applying the function first within each partition on a per-key basis and then across the partitions. While both the `groupByKey()` and `reduceByKey()` transformations can often be used to solve the same problem and will produce the same answer, the `reduceByKey()` transformation works much better for large distributed datasets. This is because Spark knows it can combine output with a common key on each partition *before* shuffling (redistributing) the data across nodes. Only use `groupByKey()` if the operation would not benefit from reducing the data before the shuffle occurs.

Look at the diagram below to understand how `reduceByKey` works. Notice how pairs on the same machine with the same key are combined (by using the lamdba function passed into reduceByKey) before the data is shuffled. Then the lamdba function is called again to reduce all the values from each partition to produce one final result.



Need to Reconstruct the List of Co-occurring Terms With the Terms of Interest

- Synchronization
  - Need to reconstruct the list of co-occurring terms with the term of interest
- Fortunately, as in the mapper, the reducer can preserve state across multiple keys
- Inside the reducer, we can buffer in memory all the words that co-occur with **wi** and their counts, in essence building the associative array in the stripes approach

Python3

```python
# Combinations Of string "GeEKS" OF SIZE 3.


from itertools import combinations

letters ="GeEKS"

# size of combination is set to 3
a = combinations(letters, 3)
y = [' '.join(i) for i in a]

print(y)
```

Output:-

```
['G e E', 'G e K', 'G e S', 'G E K', 'G E S', 'G K S', 'e E K', 'e E S', 'e K S', 'E K S']
```

https://www.geeksforgeeks.org/python-itertools-combinations-function/

Example: Bigram Language Model

<s> I am Sam </s>
<s> Sam I am </s>
<s> I do not like green eggs and ham </s>

Training Corpus

P(I | <s>) = 2/3 = 0.67
P(am | I) = 2/3 = 0.67
P(</s> | Sam)= 1/2 = 0.50
...

P(Sam | <s>) = 1/3 = 0.33
P(do | I) = 1/3 = 0.33
P(Sam | am) = 1/2 = 0.50

Bigram Probability Estimates

Note: We don't ever cross sentence boundaries

https://www.slideserve.com/altessa/n-gram-language-models

Build a bi-gram model in Spark

itertools.combinations(word_list_for_sentence, 2)
[(<s>, I), 1]
[<s>, *), 1]
[(I, am), 1]
[I, *), 1]
[(am, sam), 1]
[(am, *), 1]
.....

Input

<s> I am Sam </s>
<s> Sam I am </s>
<s> I do not like green eggs and ham </s>

Training Corpus

Output

P(I | <s>) = 2/3 = 0.67
P(am | I) = 2/3 = 0.67
P(</s> | Sam)= 1/2 = 0.50
P(Sam | <s>) = 1/3 = 0.33
P(do | I) = 1/3 = 0.33
P(Sam | am) = 1/2 = 0.50

For Pr(I|<s>) we need all the records of the form [(<s>, I), 1] and [<s>, *), 1]

- Case 1: using one reducer (single partition)
- Case 2: using two or reducers (multiple partitions)

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

```
## Get the result
data = sc.parallelize(['dog aardvark pig banana',
                       'bear zebra pig'])
```

```
## Break it down - Flat Map getting all the pairs ready (and the total too)
pair_rdd = data.flatMap(makePairs)

pair_rdd.collect()
```

```
[(('dog', '*'), 1),
 (('dog', 'aardvark'), 1),
 (('dog', '*'), 1),
 (('dog', 'pig'), 1),
 (('dog', '*'), 1),
 (('dog', 'banana'), 1),
 (('aardvark', '*'), 1),
 (('aardvark', 'pig'), 1),
 (('aardvark', '*'), 1),
 (('aardvark', 'banana'), 1),
 (('pig', '*'), 1),
 (('pig', 'banana'), 1),
 (('bear', '*'), 1),
 (('bear', 'zebra'), 1),
 (('bear', '*'), 1),
 (('bear', 'pig'), 1),
 (('zebra', '*'), 1),
 (('zebra', 'pig'), 1)]
```

```
## Helper functions used in the implementation (from Async)
def makePairs(row):
    words = row.split(' ')
    for w1, w2 in itertools.combinations(words, 2):
        yield ((w1,"*"), 1)
        yield ((w1,w2), 1)
```

pyspark.RDD.reduceByKey¶

RDD.**reduceByKey**(*func: Callable[[V, V], V], numPartitions: Optional[int] = None, partitionFunc: Callable[[K], int] = <function portable_hash>*) → pyspark.rdd.RDD[Tuple[K, V]] [source]

Merge the values for each key using an associative and commutative reduce function.

This will also perform the merging locally on each mapper before sending results to a reducer, similarly to a "combiner" in MapReduce.

Output will be partitioned with *numPartitions* partitions, or the default parallelism level if *numPartitions* is not specified. Default partitioner is hash-partition.

New in version 1.6.0.

Parameters: **func : *function***

the reduce function

numPartitions : *int, optional*

the number of partitions in new RDD

partitionFunc : *function, optional, default portable_hash*

function to compute the partition index

Returns: RDD

a RDD containing the keys and the aggregated result for each key

Reduce by key, number of partitions, and how to partition
By default partition by key
BUT one can specify a separate partition key

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

```python
## Helper functions used in the implementation (from Async)
def makePairs(row):
    words = row.split(' ')
    for w1, w2 in itertools.combinations(words, 2):
        yield ((w1,"*"), 1)
        yield ((w1,w2), 1)

def partitionByWord(x):
    return hash(x[0][0])

def calcRelFreq(row):
    row = sorted(row, key=lambda tup: (tup[0][0], tup[0][1]))
    currPair, currWord = None, None
    pairTotal, wordTotal = 0,0
    for r in list(row):
        w1, w2 = r[0][0], r[0][1]
        if w2 == "*":
            if w1 != currWord:
                wordTotal = 0
                currWord = w1
            wordTotal += r[1]
        else:
            pairTotal += r[1]
            if currPair != r[0]:
                yield (w1 + " - " + w2, pairTotal/wordTotal)
                pairTotal = 0
                currPair = r[0]
```

```python
## Get the result
data = sc.parallelize(['dog aardvark pig banana',
                       'bear zebra pig'])

pair_rdd = data.flatMap(makePairs)\
               .reduceByKey(lambda x,y: x+y,
                            partitionFunc=partitionByWord)\
               .mapPartitions(calcRelFreq, True)

pair_rdd.glom().collect()
```

```
0 [('dog - aardvark', 0.3333333333333333),
   ('dog - banana', 0.3333333333333333),
   ('dog - pig', 0.3333333333333333)],
1 [],
2 [('pig - banana', 1.0)],
3 [('zebra - pig', 1.0)],
4 [('bear - pig', 0.5), ('bear - zebra', 0.5)],
5 [('aardvark - banana', 0.5), ('aardvark - pig', 0.5)]]
```

Pr(X | dog)

6 shards

```python
## Get the result
data = sc.parallelize(['dog aardvark pig banana',
                       'bear zebra pig'])

pair_rdd = data.flatMap(makePairs)\
```

```
(('dog', 'banana'), 1)
```

```python
pair_rdd.collect()
```

```
[(('dog', '*'), 1),
 (('dog', 'aardvark'), 1),
 (('dog', '*'), 1),
 (('dog', 'pig'), 1),
 (('dog', '*'), 1),
 (('dog', 'banana'), 1),
 (('aardvark', '*'), 1),
 (('aardvark', 'pig'), 1),
 (('aardvark', '*'), 1),
 (('aardvark', 'banana'), 1),
 (('pig', '*'), 1),
 (('pig', 'banana'), 1),
 (('bear', '*'), 1),
 (('bear', 'zebra'), 1),
 (('bear', '*'), 1),
 (('bear', 'pig'), 1),
 (('zebra', '*'), 1),
 (('zebra', 'pig'), 1)]
```

RDD.glom() → pyspark.rdd.RDD[List[T]][source]

Return an RDD created by coalescing all elements within each partition into a list.

Examples

```
>>> rdd = sc.parallelize([1, 2, 3, 4], 2)
>>> sorted(rdd.glom().collect())
[[1, 2], [3, 4]]
```

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

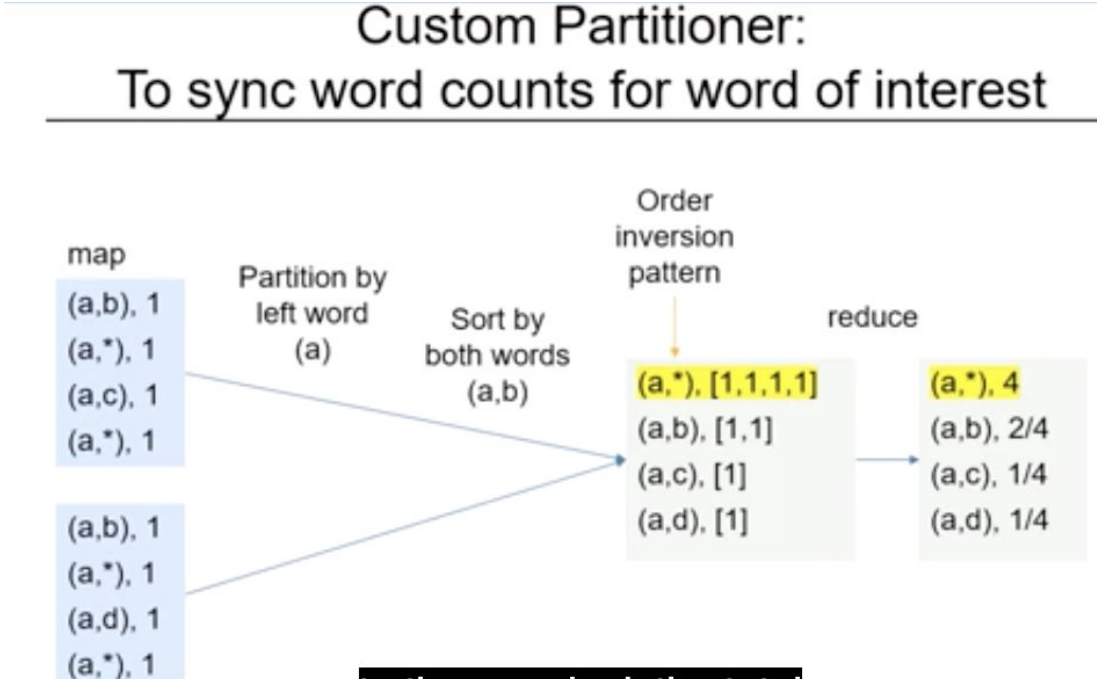
Key is a tuple but the partition key is the first element in the tuple

window((a , b, a, c, d), 2) → (a,b), (a, b), (a, c), (a, d)

Find co-occurring pairs in a doc
And calculate the relative freqs of the pairs

```
## Get the result
data = sc.parallelize(['dog aardvark pig banana',
                       'bear zebra pig'])

pair_rdd = data.flatMap(makePairs)\
```

```
[[('dog - aardvark', 0.3333333333333333),
  ('dog - banana', 0.3333333333333333),
  ('dog - pig', 0.3333333333333333)],
 [],
 [('pig - banana', 1.0)],
 [('zebra - pig', 1.0)],
 [('bear - pig', 0.5), ('bear - zebra', 0.5)],
 [('aardvark - banana', 0.5), ('aardvark - pig', 0.5)]]
```



Unit test your mapper
In python: mapper

```
## Get the result
data = sc.parallelize(['dog aardvark pig banana',
                       'bear zebra pig'])

pair_rdd = data.flatMap(makePairs)\
```

Unit test mapper to see if it produces the correct records

Order inversion?

```
##
def makePairs(row):
    """ emit all pairs of cooccurring words"""
    words = row.split(' ')
    for w1, w2 in itertools.combinations(words, 2):
        yield ((w1,"*"), 1)
        yield ((w1,w2), 1)
[x for x in makePairs('dog aardvark pig banana')]
```

```
[(('dog', '*'), 1),
 (('dog', 'aardvark'), 1),
 (('dog', '*'), 1),
 (('dog', 'pig'), 1),
 (('dog', '*'), 1),
 (('dog', 'banana'), 1),
 (('aardvark', '*'), 1),
 (('aardvark', 'pig'), 1),
 (('aardvark', '*'), 1),
 (('aardvark', 'banana'), 1),
 (('pig', '*'), 1),
 (('pig', 'banana'), 1)]
```

Pr(aardvark | dog)

In PySpark: Return all possible pairs of words in a record

```python
## Helper functions used in the implementation (from Async)
def makePairs(row):
    words = row.split(' ')
    for w1, w2 in itertools.combinations(words, 2):
        yield ((w1,"*"), 1)
        yield ((w1,w2), 1)

def partitionByWord(x):
    return hash(x[0][0])

def calcRelFreq(row):
    row = sorted(row, key=lambda tup: (tup[0][0], tup[0][1]))
    currPair, currWord = None, None
    pairTotal, wordTotal = 0,0
    for r in list(row):
        w1, w2 = r[0][0], r[0][1]
        if w2 == "*":
            if w1 != currWord:
                wordTotal = 0
                currWord = w1
            wordTotal += r[1]
        else:
            pairTotal += r[1]
            if currPair != r[0]:
                yield (w1 + " - " + w2, pairTotal/wordTotal)
                pairTotal = 0
                currPair = r[0]
```

```python
## Get the result
data = sc.parallelize(['dog aardvark pig banana',
                       'bear zebra pig'])

pair_rdd = data.flatMap(makePairs)
```

collect()

```
## Break it down - Flat Map getting all the pairs ready (and the total too)
pair_rdd = data.flatMap(makePairs)

pair_rdd.collect()
```

```
[(('dog', '*'), 1),
 (('dog', 'aardvark'), 1),
 (('dog', '*'), 1),
 (('dog', 'pig'), 1),
 (('dog', '*'), 1),
 (('dog', 'banana'), 1),
 (('aardvark', '*'), 1),
 (('aardvark', 'pig'), 1),
 (('aardvark', '*'), 1),
 (('aardvark', 'banana'), 1),
 (('pig', '*'), 1),
 (('pig', 'banana'), 1),
 (('bear', '*'), 1),
 (('bear', 'zebra'), 1),
 (('bear', '*'), 1),
 (('bear', 'pig'), 1),
 (('zebra', '*'), 1),
 (('zebra', 'pig'), 1)]
```

```python
## Helper functions used in the implementation (from Async)
def makePairs(row):
    words = row.split(' ')
    for w1, w2 in itertools.combinations(words, 2):
        yield ((w1,"*"), 1)
        yield ((w1,w2), 1)

def partitionByWord(x):
    return hash(x[0][0])

def calcRelFreq(row):
    row = sorted(row, key=lambda tup: (tup[0][0], tup[0][1]))
    currPair, currWord = None, None
    pairTotal, wordTotal = 0,0
    for r in list(row):
        w1, w2 = r[0][0], r[0][1]
        if w2 == "*":
            if w1 != currWord:
                wordTotal = 0
                currWord = w1
            wordTotal += r[1]
        else:
            pairTotal += r[1]
            if currPair != r[0]:
                yield (w1 + " - " + w2, pairTotal/wordTotal)
                pairTotal = 0
                currPair = r[0]
```

```python
## Get the result
data = sc.parallelize(['dog aardvark pig banana',
                       'bear zebra pig'])

pair_rdd = data.flatMap(makePairs)\
               .reduceByKey(lambda x,y: x+y,
                            partitionFunc=partitionByWord)\
               .mapPartitions(calcRelFreq, True)

pair_rdd.glom().collect()
```

```
0 [('dog - aardvark', 0.3333333333333333),
   ('dog - banana', 0.3333333333333333),
   ('dog - pig', 0.3333333333333333)],
1 [],
2 [('pig - banana', 1.0)],
3 [('zebra - pig', 1.0)],
4 [('bear - pig', 0.5), ('bear - zebra', 0.5)],
5 [('aardvark - banana', 0.5), ('aardvark - pig', 0.5)]]
```

6 shards

```python
## Get the result
data = sc.parallelize(['dog aardvark pig banana',
                       'bear zebra pig'])

pair_rdd = data.flatMap(makePairs)\
```

```python
pair_rdd.collect()
```

```
[(('dog', '*'), 1),
 (('dog', 'aardvark'), 1),
 (('dog', '*'), 1),
 (('dog', 'pig'), 1),
 (('dog', '*'), 1),
 (('dog', 'banana'), 1),
 (('aardvark', '*'), 1),
 (('aardvark', 'pig'), 1),
 (('aardvark', '*'), 1),
 (('aardvark', 'banana'), 1),
 (('pig', '*'), 1),
 (('pig', 'banana'), 1),
 (('bear', '*'), 1),
 (('bear', 'zebra'), 1),
 (('bear', '*'), 1),
 (('bear', 'pig'), 1),
 (('zebra', '*'), 1),
 (('zebra', 'pig'), 1)]
```

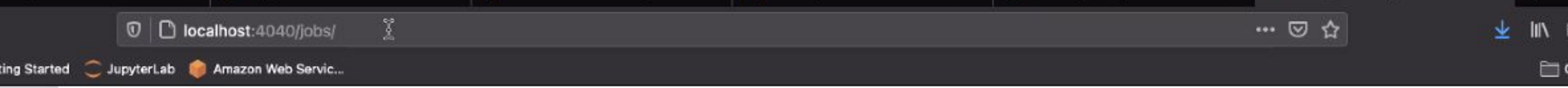
RDD.glom() →
pyspark.rdd.RDD[List[T]][source]

Return an RDD created by coalescing all elements within each partition into a list.

Examples

```
>>> rdd = sc.parallelize([1, 2, 3, 4], 2)
>>> sorted(rdd.glom().collect())
[[1, 2], [3, 4]]
```

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



Jobs | Stages | Storage | Environment | Executors | SQL | week5_demo a

(?)

larreal
n
IFO

bs (2)

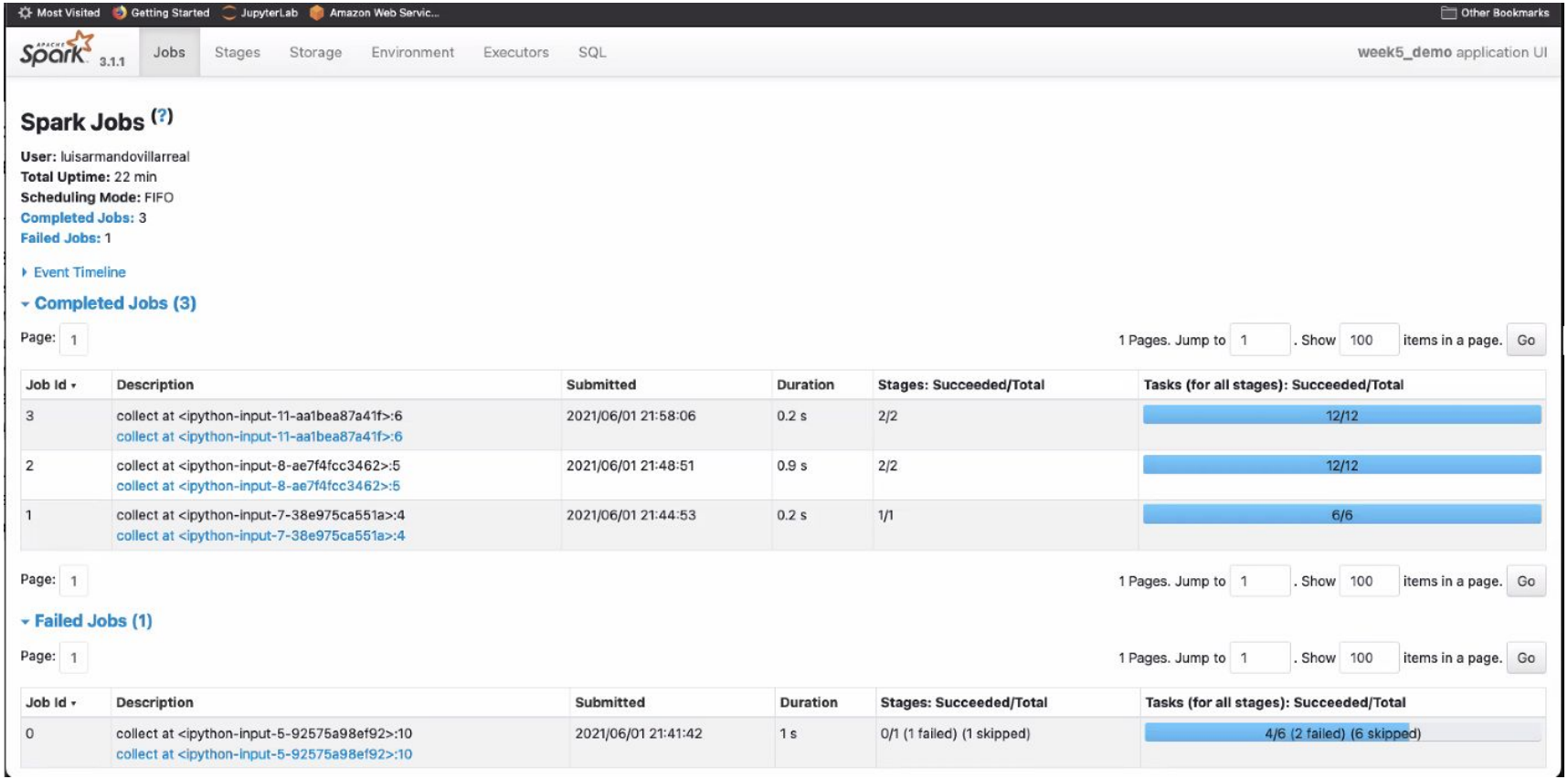
1 Pages. Jump to 1 . Show 100 items in a

ription	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
t at <ipython-input-8-ae7f4fcc3462>:5 t at <ipython-input-8-ae7f4fcc3462>:5	2021/06/01 21:48:51	0.9 s	2/2	12/12
t at <ipython-input-7-38e975ca551a>:4 t at <ipython-input-7-38e975ca551a>:4	2021/06/01 21:44:53	0.2 s	1/1	6/6

1 Pages. Jump to 1 . Show 100 items in a

1 Pages. Jump to 1 . Show 100 items in a

ription	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
t at <ipython-input-5-92575a98ef92>:10	2021/06/01 21:41:42	1 s	0/1 (1 failed) (1 skipped)	4/6 (2 failed) (6 skipped)


Most Visited
Getting Started
JupyterLab
Amazon Web Servic...
Other Bookmarks
Spark 3.1.1
Jobs
Stages
Storage
Environment
Executors
SQL
week5_demo application UI
Spark Jobs (?)
User: luisarmandovillarreal
Total Uptime: 22 min
Scheduling Mode: FIFO
Completed Jobs: 3
Failed Jobs: 1
Event Timeline
Completed Jobs (3)
Page: 1
1 Pages. Jump to 1 . Show 100 items in a page. Go
Job Id
Description
Submitted
Duration
Stages: Succeeded/Total
Tasks (for all stages): Succeeded/Total
3
collect at <ipython-input-11-aa1bea87a41f>:6
collect at <ipython-input-11-aa1bea87a41f>:6
2021/06/01 21:58:06
0.2 s
2/2
12/12
2
collect at <ipython-input-8-ae7f4fcc3462>:5
collect at <ipython-input-8-ae7f4fcc3462>:5
2021/06/01 21:48:51
0.9 s
2/2
12/12
1
collect at <ipython-input-7-38e975ca551a>:4
collect at <ipython-input-7-38e975ca551a>:4
2021/06/01 21:44:53
0.2 s
1/1
6/6
Page: 1
1 Pages. Jump to 1 . Show 100 items in a page. Go
Failed Jobs (1)
Page: 1
1 Pages. Jump to 1 . Show 100 items in a page. Go
Job Id
Description
Submitted
Duration
Stages: Succeeded/Total
Tasks (for all stages): Succeeded/Total
0
collect at <ipython-input-5-92575a98ef92>:10
collect at <ipython-input-5-92575a98ef92>:10
2021/06/01 21:41:42
1 s
0/1 (1 failed) (1 skipped)
4/6 (2 failed) (6 skipped)

```
def makePairs(row):
    words = row.split(' ')
    for w1, w2 in combinations(words, 2):
        yield((w1,"*"),1)
        yield((w1,w2),1)

def partitionByWord(x):
    return hash(x[0][0])

def calcRelFreq(row):
    row  = sorted(row, key=lambda tup: (tup[0][0], tup[0][1]))
    currPair, currWord, = None, None
    pairTotal, wordTotal = 0, 0
    for r in list(row):
        w1, w2 = r[0][0], r[0][1]
        if w2 == "*":
            if w1 != currWord:
                wordTotal = 0
                currWord = w1
            wordTotal += r[1]
        else:
            pairTotal += r[1]

            if currPair != r[0]:
                yield(w1+" - "+w2, pairTotal/wordTotal)
                pairTotal = 0
                currPair = r[0]
```

```
RDD = DATA.flatMap(makePairs)\
          .reduceByKey(add,
             partitionFunc=partitionByWord)\
          .mapPartitions(calcRelFreq, True)

RDD.glom().collect()
```

RESULT:

```
[
    [
    ('bear - pig', 0.5),
    ('bear - zebra', 0.5),
    ('dog - aardvark', 0.33),
    ('dog - banana', 0.33),
    ('dog - pig', 0.33),
    ('pig - banana', 1.0)
    ], [
    ('aardvark - banana', 0.5),
    ('aardvark - pig', 0.5),
    ('zebra - pig', 1.0)
    ]
]
```

```
DATA = sc.parallelize(['dog aardvark pig banana',
                       'bear zebra pig'])
```

Window of size 2 before and two after

The age of wisdom and foolishness is worst……

1 2 3
age wisdom foolishness worst……

Ignore stopwords like: the, of, is

	age	best	foolishness	times	wisdom	worst
age	0	0	1	0	1	0
best	0	0	0	1	0	0
foolishness	1	0	0	0	0	0
times	0	1	0	0	0	1
wisdom	1	0	0	0	0	1
worst	0	0	0	1	0	0

In the context of hw3...

③ inference : Which word pairs are most similar in meaning?

eg. apple ≠ boy
apple ≠ orange
boy ≠ orange

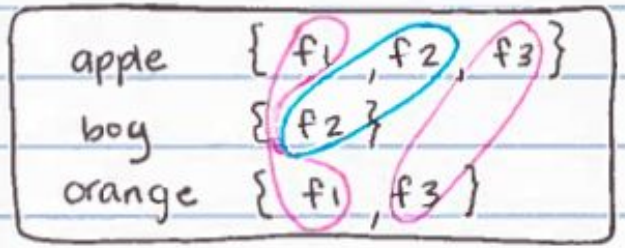
② model : compute similarity score for each pair.

① features: binary "co-occurrance" w/ BASIS words.

eg. apple co-occurs with "tree"
co-occurs with "pie"
co-occurs with "red"
does NOT co-occur w/ "child"


"coocurs with tree"
Thinking through data structure ...
f1 f2 f3
apple 1 1 1
boy 0 1 0
orange 1 0 1
apple { f1, f2, f3 }
boy { f2 }
orange { f1, f3 }
matrix vs stripes

Computing "similarity"

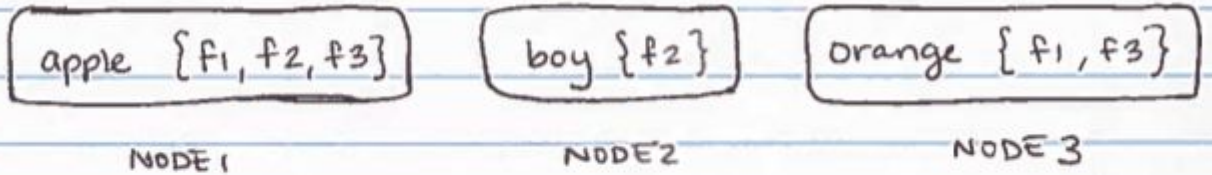


apple + boy : 1

apple + orange : 2

orange + boy : 0

Could we do this "in parallel"?



apple + boy
?

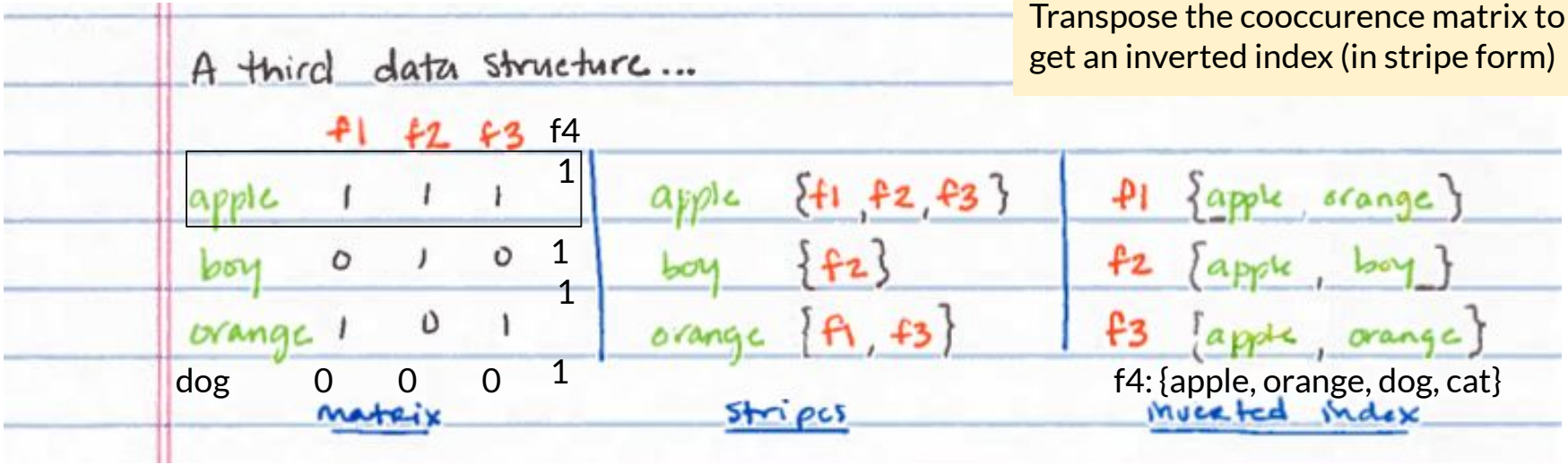
apple + orange
?

orange + boy
?

Represent co-occurrence information in different ways

n^2 problem

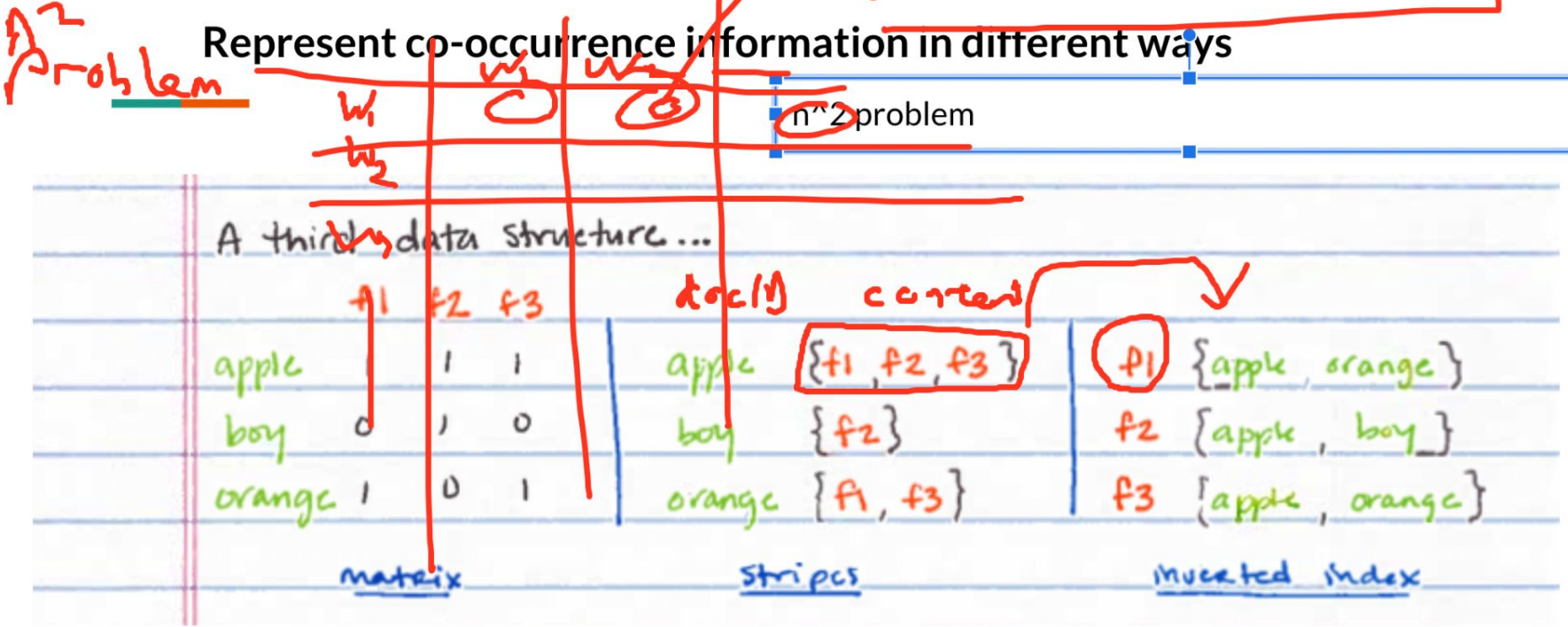
Transpose the cooccurence matrix to to get an inverted index (in stripe form)



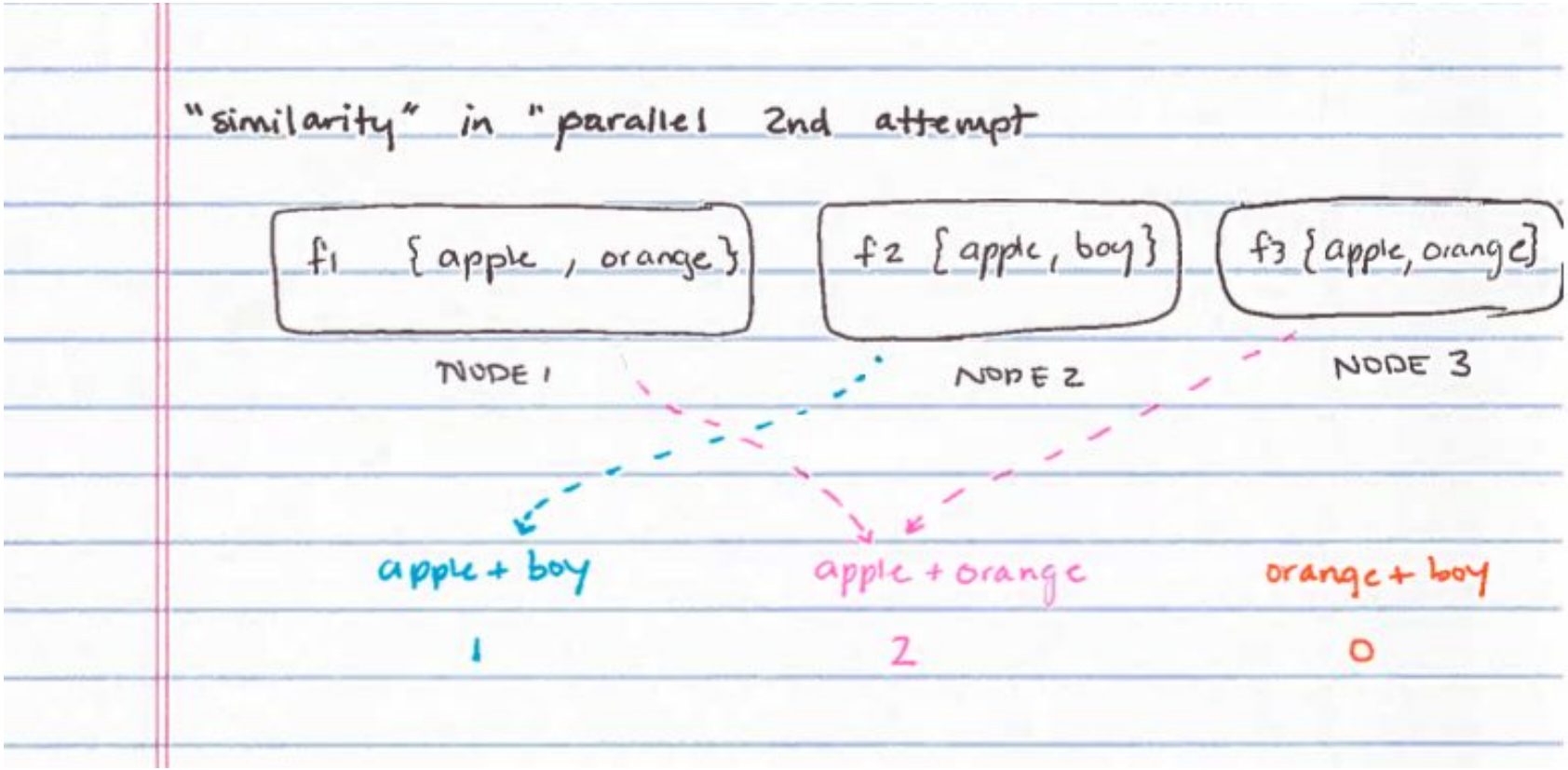
Sim(apple, orange) = ⅔ #jaccard
Sim(boy, orange) = 0/3 #jaccard; ¼ in case of adding in f4

Represent co-occurrence information in different ways

n^2 problem



Sim(apple, orange) = ⅔ #jaccard
Sim(boy, orange) = 0/3 #jaccard


"similarity" in "parallel 2nd attempt
f1 {apple, orange}
f2 {apple, boy}
f3 {apple, orange}
NODE 1
NODE 2
NODE 3
apple + boy
apple + orange
orange + boy
1
2
0

Window of size 2 before and two after

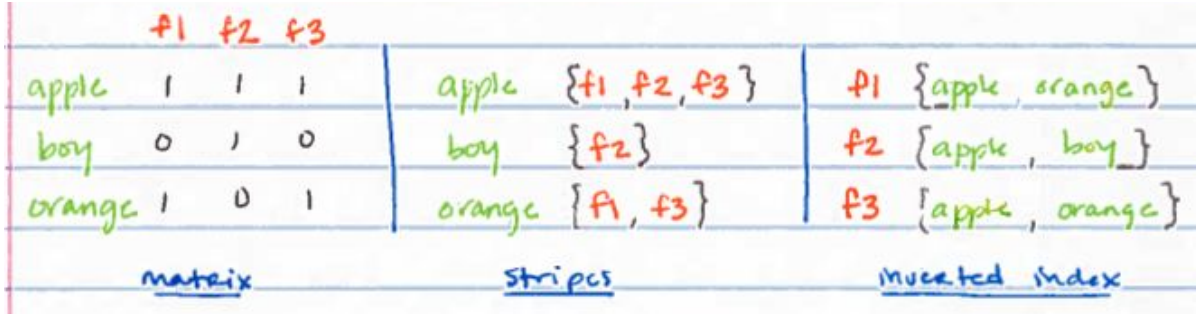
The age of wisdom and foolishness is worst……

1 2 3
age wisdom foolishness worst……

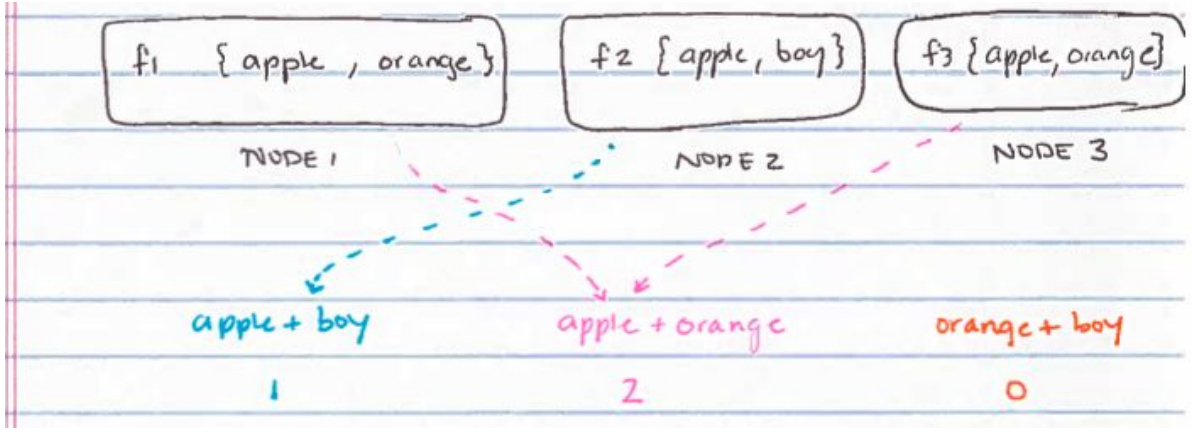
Ignore stopwords like: the, of, is

	age	best	foolishness	times	wisdom	worst
age	0	0	1	0	1	0
best	0	0	0	1	0	0
foolishness	1	0	0	0	0	0
times	0	1	0	0	0	1
wisdom	1	0	0	0	0	1
worst	0	0	0	1	0	0

Find similarity between two terms based on their concurrence vectors



Use the inverted index to scale the term similarity calculation



Checkpoint the full stripes for each term to disk. Repartition data first and then save (as otherwise, we will end up with 190 partitions; I wonder why!).

10.0.3.1. Checkpoint the stripes (Just RUN all cells in this section AS IS and review outputs)

Let's save your full stripes to disk. Then we can reload later if needed. We reparition our data first and then save (as otherwise, we will end up with 190 partitions; I wonder why!).

```
!gsutil -m rm -r {HW3_FOLDER}/stripes 2> /dev/null   ##remove old results
stripesRDD.repartition(4).saveAsTextFile(f'{HW3_FOLDER}/stripes')  #repartition and write partitions to Google Cloud Bucket
!gsutil ls -lh {HW3_FOLDER}/stripes
```

The above produces the following output directory:

```
       0 B  2022-05-31T05:35:02Z  gs://w261-jgs/main/Assignments/HW3/docker/student/stripes/
       0 B  2022-05-31T05:35:03Z  gs://w261-jgs/main/Assignments/HW3/docker/student/stripes/_SUCCESS
  1.74 MiB  2022-05-31T05:35:02Z  gs://w261-jgs/main/Assignments/HW3/docker/student/stripes/part-00000
   1.6 MiB  2022-05-31T05:35:01Z  gs://w261-jgs/main/Assignments/HW3/docker/student/stripes/part-00001
  1.58 MiB  2022-05-31T05:35:02Z  gs://w261-jgs/main/Assignments/HW3/docker/student/stripes/part-00002
  1.52 MiB  2022-05-31T05:35:01Z  gs://w261-jgs/main/Assignments/HW3/docker/student/stripes/part-00003
TOTAL: 6 objects, 6745577 bytes (6.43 MiB)
```

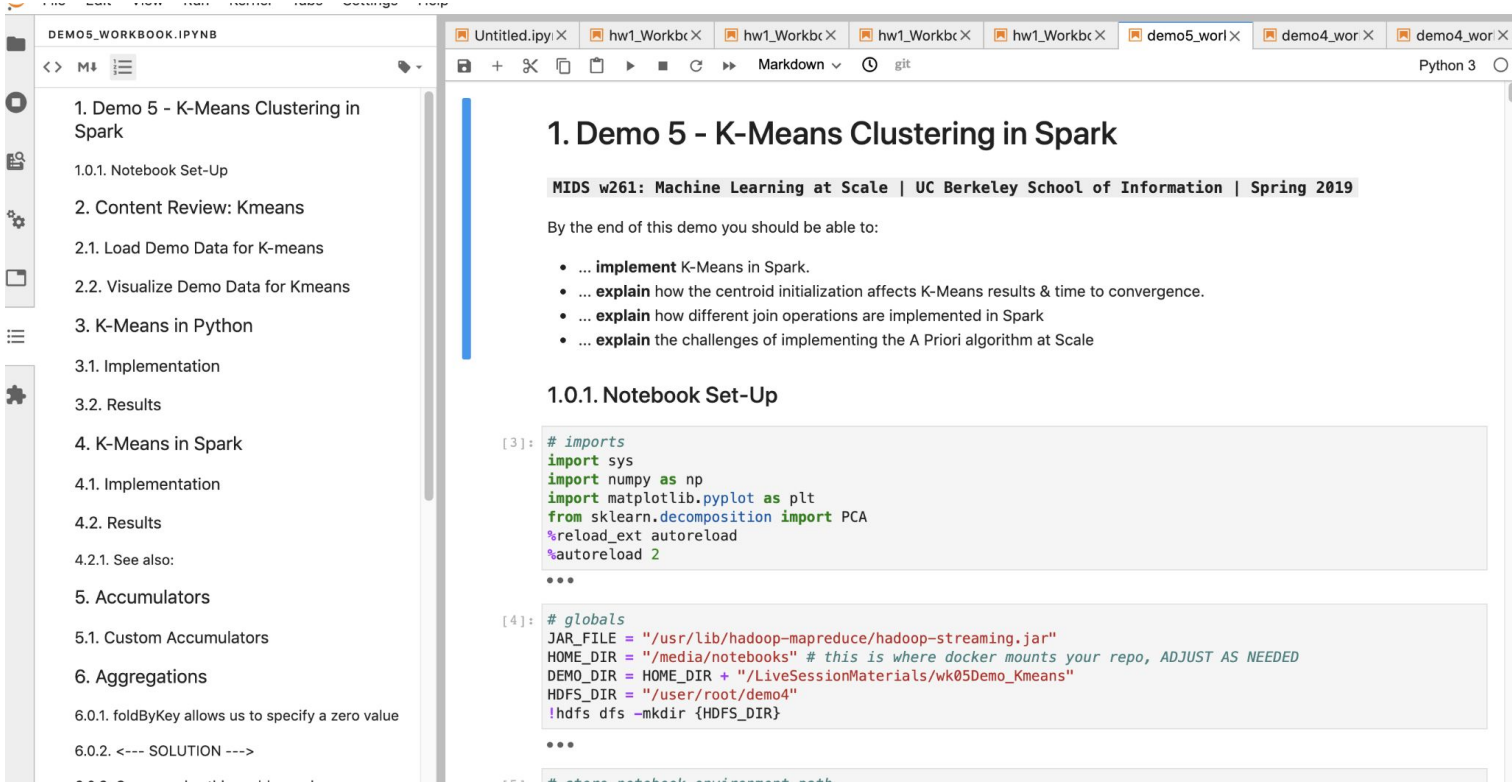
The following code displays the stripe of cooccurence words for the term `sea`:

```
!gsutil cat {HW3_FOLDER}/stripes/part-00000|head -n 1
```

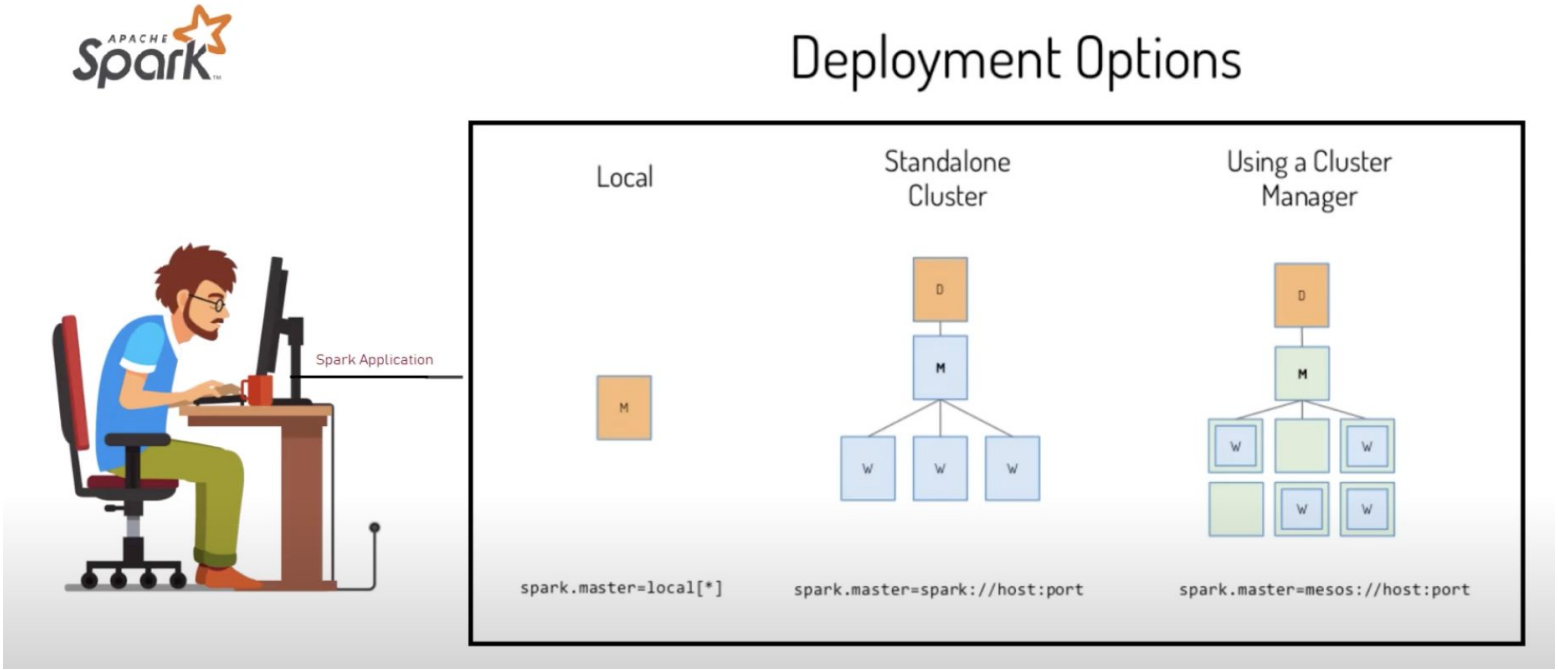
```
('sea', {'sweeping', 'twisted', 'athenians', 'fog', 'tumult', 'repression', 'morphology', 'jane', 'secreted', 'tents', 'barred', 'sadness', 'hamlet', 'turbulent', 'rains',
'robe', 'imagery', 'myths', 'orient', 'intervening', 'victories', 'accumulate', 'sinners', 'constancy', 'strained', 'sermons', 'shoe', 'trembled', 'merged', 'eastward',
'avoidance', 'sensibility', 'informing', 'silently', 'dip', 'surround', 'blocked', 'voyages', 'bursting', 'vastly', 'southeastern', 'cracks', 'tore', 'temperament',
"ship's", 'odor', 'atlas', 'matthew', 'ether', 'colonization', 'irresistible', 'shells', 'alaska', 'gaza', 'distributions', 'farthest', 'silly', 'flush', 'ugly',
'transparent', 'arabian', 'sandy', 'steering', 'penetrating', 'burns', 'norway', 'thames', 'moonlight', 'plunge', 'beset', 'yielding', 'tuesday', 'impacts', 'cheese',
'convex', 'armistice', 'polished', 'freshness', 'belgium', 'saturation', 'dumb', 'spoil', 'shines', 'sunset', 'softly', 'laden', 'realms', 'alexandria', 'parallels',
'weep', 'ushered', 'violently', 'expanse', 'travellers', 'insoluble', 'downs', 'roofs', 'filtered', 'ashore', 'graces', 'obscured', 'establishments', 'traversed',
'crystalline', 'warmer', 'skins', 'viewing', 'fascination', 'liverpool', 'contamination', 'sails', 'masculine', 'usages', 'bucket', 'dipped', 'dew', 'fare', 'overlooking',
'necks', 'sticks', 'weighing', 'danube', 'mast', 'phosphorus', 'mate', 'attested', 'anonymous', 'wax', 'finishing', 'parked', 'flocks', 'humidity', 'endurance', 'terrors',
'carpet', 'misfortunes', 'hydroxide', 'crazy', 'priesthood', 'hungary', 'nova', 'believeth', 'remotest', 'occupants', 'complexion', 'floors', 'stationary', 'provoked',
'osmotic', 'spoils', 'clearance', 'hangs', 'openings', 'halfway', 'inorganic', 'nursery', 'vigilance', 'conqueror', 'ft', 'feathers', 'roses', 'emblem', 'lawn', 'damp',
'switzerland', 'drinks', 'contradictory', 'drained', 'ordinances', 'captains', 'barren', 'steamer', 'pursuits', 'storms', 'wasting', 'frankly', 'sequences', 'pitched',
'aggravated', 'viceroy', 'leaped', 'cunning', 'simon', 'marching', 'lends', 'sherman', 'centered', 'genome', 'iran', 'sued', 'imputed', 'perilous', 'desperately',
'southward', 'maiden', 'unusually', 'crosses', 'revealing', 'uppermost', 'remission', 'inherit', 'sunny', 'ink', 'restless', 'lighting', 'serpent', 'scarlet', 'hebrews',
'flourish', 'terminology', 'bidding', 'autobiography', 'despise', 'signification', 'preparatory', 'radioactive', 'drying', 'persia', 'unfamiliar', 'twist', 'fiery',
'boon', 'delights', 'commonest', 'bounty', 'traders', 'whoever'})
```

KMeans

Goto notebook: kmeans in python and in PySpark



Container runs Spark in local mode



Local mode

- Local Mode also known as Spark in-process is the default mode of spark. It does not require any resource manager. It runs everything on the same machine. Because of local mode, we are able to simply download spark and run without having to install any resource manager.
- With local mode, we can utilize multiple core of a CPU for processing. Essentially, It is good for parallel computing.
- Since the smallest unit of parallelization is a partition, **the partitions are generally kept less than or equal to number of CPUs available.** If we keep partitions more than the CPUs, it would not give any additional advantage with respect to parallelization.
- The local mode is also quite useful while testing a spark application.

Spark Logs

- Using Python's `logging` module
- Messages are written to **stderr** when running a local mode application
- Messages are written to **log files** by your cluster manager when running Spark on a cluster

```
spark.sparkContext.setLogLevel("INFO")
```

```
import logging

logging.warning('Watch out!')  # will print a message to the console

logging.info('I told you so')  # will not print anything
```

Logging HOWTO

Legacy slides

Estimate cooccurance

We want to estimate the the relative frequency of tokens in the following text:

TEXT: A D CE A D E B A C E D

1. Count the co-occurances in windows of 2.
2. Count the total frequency of each term.

Count(X,Y)	
A,B	1
A,C	1
A,D	2
A,E	1
B,E	1
C,D	1
C,E	2
D,E	2

Count(A) = $\sum_{B'}$Count(A,B')	
A	5
B	2
C	4
D	5
E	6

$$f(B \mid A) = \frac{\text{count}(A, B)}{\text{count}(A)} = \frac{\text{count}(A, B)}{\sum_{B'} \text{count}(A, B')}$$

In order to calculate $f(\mathrm{B}|\mathrm{A})$ the reducers need

- count(A,B)
  &
- count(A)

Q: What is the most efficient way to do this in Map Reduce?

“Order Inversion” enables stateless reducers

- **Common design pattern**
  - Computing relative frequencies requires marginal counts
  - But marginal cannot be computed until you see all counts
  - **Buffering (stateful) is a bad idea!**
  - Trick: getting the marginal counts to arrive at the reducer before the joint counts
- **Optimizations**
  - Apply in-memory combining pattern to accumulate marginal counts
  - Should we apply combiners?

Order Inversion Pattern in the stream

- It is so named because through proper coordination, we can access the result of a computation in the reducer (for example, an aggregate statistic) before processing the data needed for that computation.
- The key insight is to convert the sequencing of computations into a sorting problem.

KEY

Get the Word counts counts out first (as they will be denominator for relative frequency)

key	values	
(dog, ∗)	[6327, 8514, . . .]	compute marginal: $\sum_{w'} N(\text{dog}, w') = 42908$
(dog, aardvark)	[2,1]	$f(\text{aardvark}\|\text{dog}) = 3/42908$
(dog, aardwolf)	[1]	$f(\text{aardwolf}\|\text{dog}) = 1/42908$
. . .		
(dog, zebra)	[2,1,1,1]	$f(\text{zebra}\|\text{dog}) = 5/42908$
(doge, ∗)	[682, . . .]	compute marginal: $\sum_{w'} N(\text{doge}, w') = 1267$
. . .		

Figure 3.12: Example of the sequence of key-value pairs presented to the reducer in the pairs algorithm for computing relative frequencies. This illustrates the application of the order inversion design pattern.

Order Inversion Example

TEXT: A D CE A D E B A C E D

Count(X,Y)	
A,B	1
A,C	1
A,D	2
A,E	1
B,E	1
C,D	1
C,E	2
D,E	2

Order inverted yeilds	
A,*	{1,1,2,1}
B,*	{1,1}
C,*	{1,1,2}
D,*	{2,1,2}
E,*	{1,1,2,2}
A,B	1
A,C	1
A,D	2
A,E	1
B,E	1
C,D	1
C,E	2
D,E	2

$$f(B \mid A) = \frac{\text{count}(A,B)}{\text{count}(A)} = \frac{\text{count}(A,B)}{\sum_{B'} \text{count}(A,B')}$$

Strategic inversion

In order to generate $\text{count}(A)$ each mapper has 2 yeild key types:

1. (A,*)
2. (A,B), (A,C), (A,D), (A,E)

- With (A,*) key is sorted and sent to the A reducers first with a secondary sort,
  - the reducers can calculate $\text{count}(A)$, and…
  - then calculate the individual $\text{count}(A,B)$ values to calculate $f(B|A)$

Data Representations: Dense vs Pairs vs Stripes

DENSE REPRESENTATION

	A	B	C	D	E
A	0	1	3	2	3
B	1	0	1	0	1
C	3	1	0	2	2
D	2	0	2	0	4
E	3	1	2	4	0

PAIRS REPRESENTATION

A,B	1
A,C	3
A,D	2
A,E	3
B,C	1
B,E	1
C,D	2
C,E	2
D,E	4

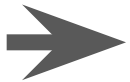
SPARSE REPRESENTATION (AKA "Stripes")

A	{B:1,C:3,D:2,E:3}
B	{A:1,C:1,E:1}
C	{A:3,B:1,D:2,E:2}
D	{D:2,C:2,E:4}
E	{A:3,B:1,C:2,D:4}

“Stripe” efficiency

- **Idea: group together pairs into an associative array**

PAIRS REPRESENTATION	
A,B	1
A,C	3
A,D	2
A,E	3



SPARSE REPRESENTATION (AKA “Stripes”)	
A	{B:1,C:3,D:2,E:3}

- **Each mapper takes a sentence:**
  - G pairs
  - Fo ch term, emit a → { b: count$_b$, c: count$_c$, }
- **Redu s perform element-wise sum of as ciative arrays**

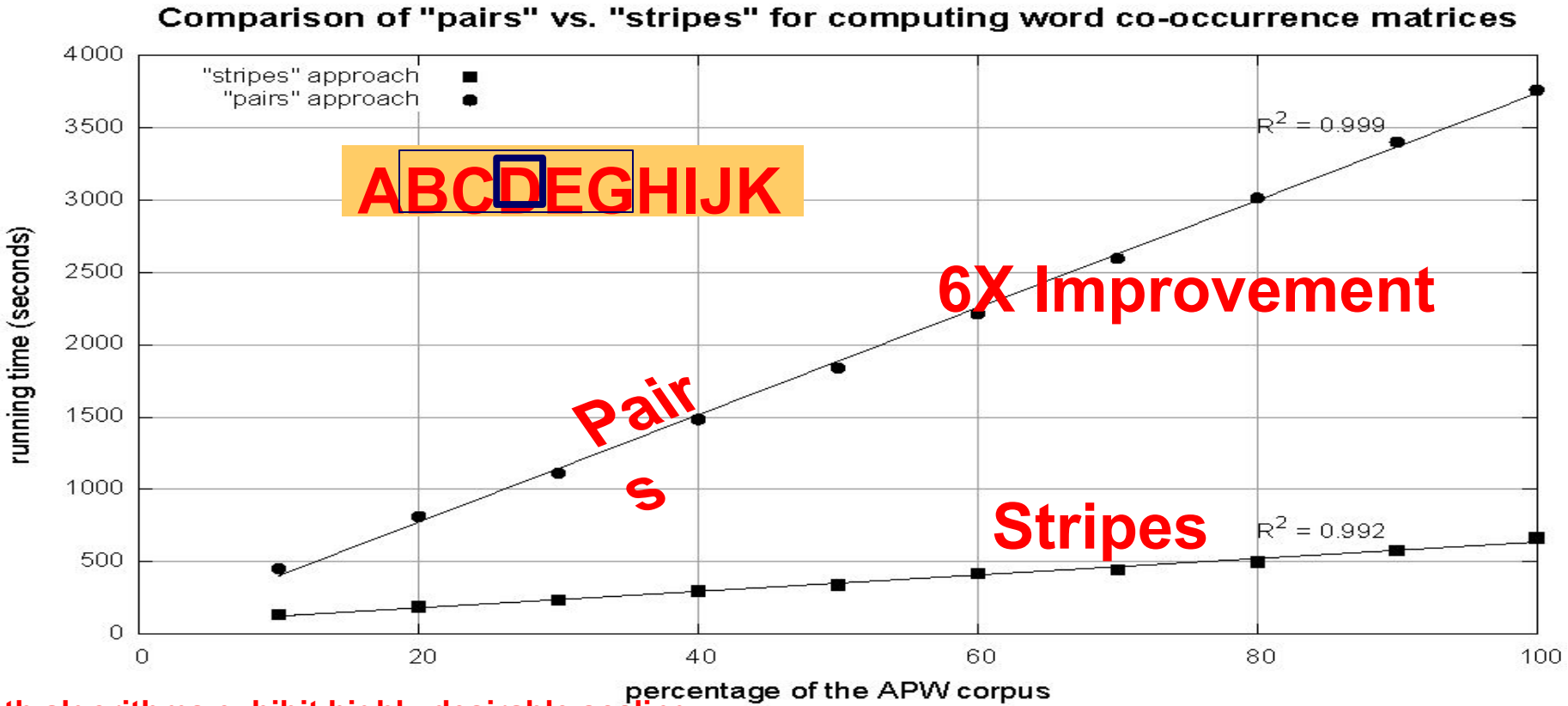
partial stripes	
A	{B:1,C:1,D:1 }
A	{B:1,D:1,E:1}
A	{ C:1, E:1}

SPARSE REPRESENTATION (AKA “Stripes”)	
A	{B:1,C:3,D:2,E:3}

Stripes are more efficient

- **Stripes A: {b1:count, ….b10: count}**
  - Transmit once per streamed text string
  - Sort

- **Pairs (a, b1, count); (a, b2, count); …(a, b10, count)**
  - Sorting is 10X
  - (a, b1, count)
  - (a, b1, count)
  - (a, b1, count)



Both algorithms exhibit highly desirable scaling

Cluster size: 38 cores (19 slave nodes with two cores)

Data Source: Associated Press Worldstream (APW) of the English Gigaword Corpus (v3), which contains 2.27 million documents (1.8 GB compressed, 5.7 GB uncompressed)

Stripes lead to a 6X improvement in terms of CPU time on entire corpus

Description	Pairs: KV pairs are "word pairs"	Stripes: KV pairs are stripes
CPU time on 38 cores	62 Minutes	11 Minutes
Mappers	2.6 Billion KV pairs, i.e., "pairs" (32GB)	653M KV Pairs (48GB), i.e., stripes
After Combiners	1.1B KV pairs, i.e., "pairs" (~14GB est.)	29M KV Pairs, i.e., stripes (~10 Gig estimated)
Co-occurrence matrix	142M word pairs (on average 100 words co-occur, i.e.,142M pairs/ 1.69M words)	**1.69 million final key-value pairs (the number of rows in the co-occurrence matrix)**

- **PAIRS**
  - The mappers in the pairs approach generated 2.6 billion intermediate key-value pairs totaling 31.2 GB. After the combiners, this was reduced to 1.1 billion key-value pairs, which quantifies the amount of intermediate data transferred across the network. In the end, the reducers emitted a total of 1.42 million final key-value pairs (the number of non-zero cells in the co-occurrence matrix).
- **Stripes**
  - Mappers emitted 653M KV Pairs (i.e., stripes) (48GB) but after combiners this reduced to 29Million stripes (extrapolate that the amount of data transferred from mappers to reducers is less 10GB)
  - As expected, the stripes approach provided more opportunities for combiners to aggregate